9-19-2024 10:30 AM

Continual Learning via Hessian-Aware Low-Rank Perturbation

Jiaqi Li, *Western University*

Supervisor: Boyu Wang, *The University of Western Ontario*
Co-Supervisor: Charles X. Ling, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

Continual learning aims to learn a series of tasks sequentially without forgetting the knowledge acquired from the previous ones. In this work, we propose the Hessian-Aware Low-Rank Perturbation (HALRP) algorithm for continual learning. By modeling the parameter transitions along the sequential tasks with the weight matrix transformation, we propose to apply the low-rank approximation on the task-adaptive parameters in each layer of the neural networks. Specifically, we theoretically demonstrate the quantitative relationship between Hessian information and the proposed low-rank approximation. The approximation ranks are then globally determined according to the marginal change of the empirical loss estimated by the layer-specific gradient and low-rank approximation error. Furthermore, we control the model capacity by pruning the less important parameters to diminish the parameter growth. We conduct extensive experiments on various benchmarks, including a dataset with large-scale tasks, and compare our method against some recent state-of-the-art methods to demonstrate the effectiveness and scalability of our proposed method. Empirical results show that our method performs better on different benchmarks, especially in achieving task order robustness and handling the forgetting issue. The source code is at https://github.com/lijiaqi/HALRP.

Keywords non-stationary environment, continual learning, task-incremental learning, Hessian matrix, low-rank approximation

Summary for Lay Audience

Unlike the conventional machine learning paradigm that assumes that there is merely one task and the training data are provided simultaneously for learning, continual learning focuses on practical scenarios where the data can be collected from different tasks and will be learned sequentially. Naturally, the fundamental objective in continual learning is to achieve good performance in the new task and keep the ability gained from existing ones, usually termed the *stability-plasticity dilemma*.

To overcome the *catastrophic forgetting* issue on previous tasks, task-specific parameters are usually introduced for each new task to isolate the learned knowledge among tasks. However, this parameter isolation strategy usually leads to a significant increase in model size when learning more tasks. Thus, a trade-off between the overall performance and the *model growth control* should necessarily be considered. Furthermore, some previous studies also showed that when facing different orders of the same set of tasks, a learner sometimes fails to guarantee consistent performance for each individual task, raising concerns with respect to the *task-order robustness* in continual learning.

In this work, we proposed the Hessian-Aware Low-Rank Perturbation (HALRP) for continual learning. Specifically, we model the parameter transition among tasks under the form of residual matrix transformation, then the low-rank approximation on the task-adaptive parameters in each layer of a neural network. With theoretical support, we show the relationship between the Hessian matrix and the low-rank approximation error. We proposed determining the approximation rank in each layer according to the marginal change of the empirical loss. Thus, a better trade-off between overall performance and model size growth can be achieved. Furthermore, by adopting the residual form for the weight transformation, our proposed method is more robust on different task orders. Experiments on common datasets and network architectures were conducted to demonstrate the effectiveness of our method.

Co-Authorship Statement

This thesis is mainly based on one paper co-authored with my supervisors, Dr. Charles X. Ling and Dr. Boyu Wang:

Hessian Aware Low-Rank Perturbation for Order-Robust Continual Learning.
by **Jiaqi Li**, Yuanhao Lai, Rui Wang, Changjian Shui, Sabyasachi Sahoo, Charles X. Ling, Shichun Yang, Boyu Wang, Christian Gagné, Fan Zhou. In IEEE Transactions on Knowledge and Data Engineering. doi: 10.1109/TKDE.2024.3419449.

I certify that I am the lead author of this article by developing the theory and performing the experiments. The credit authorship contribution statement of this published paper is summarized as follows:

Jiaqi Li: Conceptualization, Methodology, Software, Validation, Writing (original draft).
Yuanhao Lai: Conceptualization.
Rui Wang: Computational resources.
Changjian Shui: Computational resources.
Sabyasachi Sahoo: Validation.
Charles X. Ling: Funding acquisition, Supervision.
Shichun Yang: Computational resources.
Boyu Wang: Funding acquisition, Supervision.
Christian Gagné: Supervision, Writing (review & editing), Funding acquisition.
Fan Zhou: Supervision, Validation, Writing (review & editing).

Table of Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

1 Introduction

1.1 Motivation

The conventional machine learning paradigm assumes all the data are simultaneously accessed and trained. However, in practical scenarios, data are often collected from different tasks and sequentially accessed in a specific order. Continual Learning (CL) aims to gradually learn from novel tasks and preserve valuable knowledge from previous ones. Despite this promising paradigm, CL is usually faced with a dilemma between *memory stability* and *learning plasticity*: when adapting among the dynamic data distributions, it has been shown that the neural networks can easily forget the learned knowledge of previous tasks when facing a new one, which is known as *Catastrophic Forgetting* (CF) [4].

One possible reason for this forgetting issue can be the parameter drift from the previous tasks to the new ones, which is caused by the optimization process with the use of stochastic gradient descent and its variants [5, 6]. To mitigate the obliviousness to past knowledge, some works [7–11] proposed to constrain the optimization objective for the knowledge of new tasks with additional penalty regularization terms. These approaches were shown as ineffective under the scenarios of a large number of tasks, lacking long-term memory stability in real-world applications.

To address this problem, some methods choose to expand the model during the dynamic learning process in a specific way, leading to inferencing with task-specific parameters for each task. For example, a recent work [2] proposed to decompose the model as task-private and task-shared parameters via an additive model parameters decomposition. However, this method only applies an attention vector for the masking of the task-shared parameters. Furthermore, the private parameters were controlled by a regularization and consolidation

process, which lead to the linear model increment along with the increasing task number, damaging the *scalability* for deploying a CL system. Another solution to reduce the parameter increase is to apply factorization for the model parameters. In this regard, [12] proposed a low-rank factorization method for the model parameters decomposition with a Bayesian process inference. However, this approach required a large rank for achieving desirable accuracy and also suffered from ineffectiveness in some complex data scenarios.

In this work, we propose a low-rank perturbation method to learn the relationship between the learned model (base parameters) and the parameters for new tasks. By formulating a flexible weight transition process, the model parameters during the CL scenario are decomposed as task-shared ones and task-adaptive ones. The former can be adopted from the base task to new tasks. The latter conforms to a low-rank matrix, and the number of relevant parameters can be effectively reduced for every single layer in a neural network across the tasks. With a simple warm-up training strategy, low-rank task adaptive parameters can be efficiently initialized with *singular value decomposition*.

Furthermore, to determine which ranks should be preserved for the low-rank approximation in different layers in a model, we propose to measure the influences of the introduced low-rank parameters by the *Hessian-aware risk perturbation* across layers of the whole model. This allows the model to automatically assign larger ranks to a layer that contributes more to the final performance under a specific parameter size budget. We theoretically support this approach by providing a formal demonstration that the empirical losses derived from the proposed low-rank perturbation are bounded by the Hessian and the approximation rate of the decomposition. This leads to a Hessian-aware framework that enables the model to determine perturbation ranks that minimize the overall empirical error. Lastly, we apply a pruning technique to control the size of introduced parameters and reduce some less important perturbation parameters through the aforementioned Hessian-aware framework

by evaluating their importance.

Based on the theoretical analysis, we proposed the **H**essian **A**ware **L**ow-**R**ank **P**erturbation (HALRP) algorithm, which enables the model to leverage the Hessian information to automatically apply the low-rank perturbation according to a certain approximation rate.

1.2 Contribution Summary

To summarize, the contributions of our work mainly lie in three-fold:

- We proposed a Hessian-Aware Low-Rank Perturbation framework that allows efficient memory and computation requests for learning from continual tasks. Based on a residual learning strategy for the parameter transition process, the performance robustness can be better achieved under different task orders.
- Theoretical analysis showed that Hessian information can be used to quantitatively measure the influences of low-rank approximation on the empirical risk. This leads to an automatic rank selection process to control the model increment.
- Extensive experiments on common benchmarks and comparisons to recent state-of-the-art baselines (*e.g.*, regularization-based, expansion-based, and rehearsal-based) were conducted to demonstrate the effectiveness of our method. The results testify the superiority of our HALRP, in terms of accuracy, computational efficiency, scalability, and task-order robustness.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 described the research background of this thesis, including the preliminary knowledge of continual learning and the challenges in this area, the basic knowledge of low-rank approximation for matrices, and a

brief overview of existing studies related to the research topic in this thesis. In Chapter 3, we described our proposed method HALRP in detail, including the parameter transition modeling between tasks for continual learning, the low-rank approximation on the parameters, the rank selection method for parameter approximation through Hessian information, and then a summary of the proposed algorithm. In Chapter 4, we provided extensive empirical verifications to demonstrate the effectiveness of our proposed method compared to the existing baseline methods, including the performance on different datasets with different network architectures, as well as the superiority on different aspects like the robustness with different task orders, the comparisons of catastrophic forgetting, computational efficiency, etc. Chapter 5 summarized this thesis and discussed the potential future work. Furthermore, in Appendices A-E, we attached some supplementary materials, including the proofs of some contents in the main text, additional experimental results, the detailed setting of our experiments, etc.

Chapter 2

2 Background

2.1 Preliminary

This section introduces the basic knowledge of continual learning and singular value decomposition. In Table 2.1, we provide a list of the main notations used in this thesis. Specifically, the bold uppercase (or lowercase) letters indicate the matrices (or vectors) in the remaining part.

Table 2.1: List of main notations.

Notation	Description
T,t	Total task number, task index
$\mathcal{T}_t, \mathcal{D}_t$	t-th task, and the related dataset
$\mathbf{x}, y$	input/sample, class label
f	the model in continual learning, i.e., neural networks
$\mathbf{W}$	model weights
$\mathbf{R}, \mathbf{S}, \mathbf{B}$	task private model weights
$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$	SVD decomposition on $\mathbf{B}$ shown in Sect. 2.1.3
diag($\mathbf{\Sigma}$)	the set of diagonal elements of $\mathbf{\Sigma}$
Diag($\sigma_1, \cdots, \sigma_n$)	diagonal matrix with elements $\{\sigma_1, \cdots, \sigma_n\}$ as the diagonal
$\mathbf{M}^{(k)}$	low-rank approximation of $\mathbf{M}$ by keeping k ranks
$\|\cdot\|_F$	Frobenius norm
$\mathcal{L}$	loss function for the task (e.g, cross-entropy)
$\mathcal{L}_{\text{reg}}$	regularization loss
$\mathbf{H}$	Hessian matrix
α	loss approximation rate for rank selection
l	layer index in a neural network
$\mathbf{g}_l$	gradient vector for layer l
r_l	full rank of layer l
k_l	approximated rank for layer l
n	number of total training epochs
n_w	number of warm-up epochs

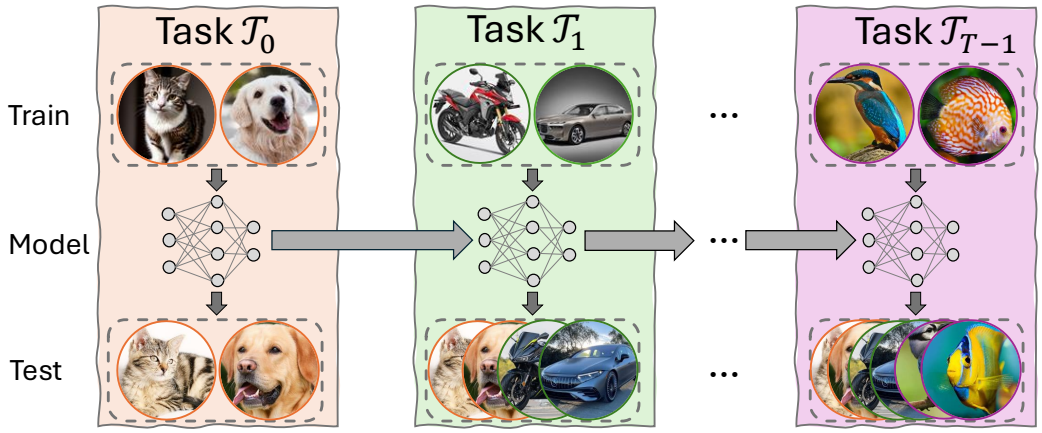


Figure 2.1: Continual Learning: learn from a sequence of tasks $\{\mathcal{T}_0, \mathcal{T}_1, \cdots, \mathcal{T}_{T-1}\}$ with a single model

2.1.1 Continual Learning: Problem Formulation

In continual learning (CL), the model (i.e., neural networks) receives a series of T tasks (i.e., usually classification tasks) $\{\mathcal{T}_0, \ldots, \mathcal{T}_{T-1}\}$ sequentially. Assume we have a training dataset for the t-th task: $\mathcal{D}_t = \{\mathbf{x}_t^i, y_t^i\}_{i=1}^{N_t} \sim \mathcal{T}_t$, where $\mathbf{x}_t^i$ and y_t^i correspond to the i-th instance and the corresponding label within the total of N_t data points. Denoting the input space as $\mathcal{X}$ and the label space of t-th task $\mathcal{T}_t$ as $\mathcal{Y}_t$. After finishing $b+1$ tasks (i.e. $\mathcal{T}_0, \ldots, \mathcal{T}_b$ with $0 \leq b \leq T-1$) in the learning process, the overall objective is to find a model (i.e., neural networks) $f : \mathcal{X} \longmapsto \mathcal{Y}_0 \cup \mathcal{Y}_1 \cup \cdots \cup \mathcal{Y}_b$ such that it can achieve the best performance on all currently seen tasks:

$$\min_f \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{T}_0\cup\cdots\cup\mathcal{T}_b}\Big[\mathbb{I}\{y \neq f(\mathbf{x})\}\Big] \tag{2.1}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function[1].

Based on the relationship among the label spaces $\mathcal{Y}_0, \ldots, \mathcal{Y}_{T-1}$ of different tasks and the knownness of task identity during training and testing, continual learning is mainly catego-

[1]Here, we adopt the form of classification task that most of the existing work considered. Thus, the best performance can be represented by the minimum classification error across all tasks.

rized as three scenarios:

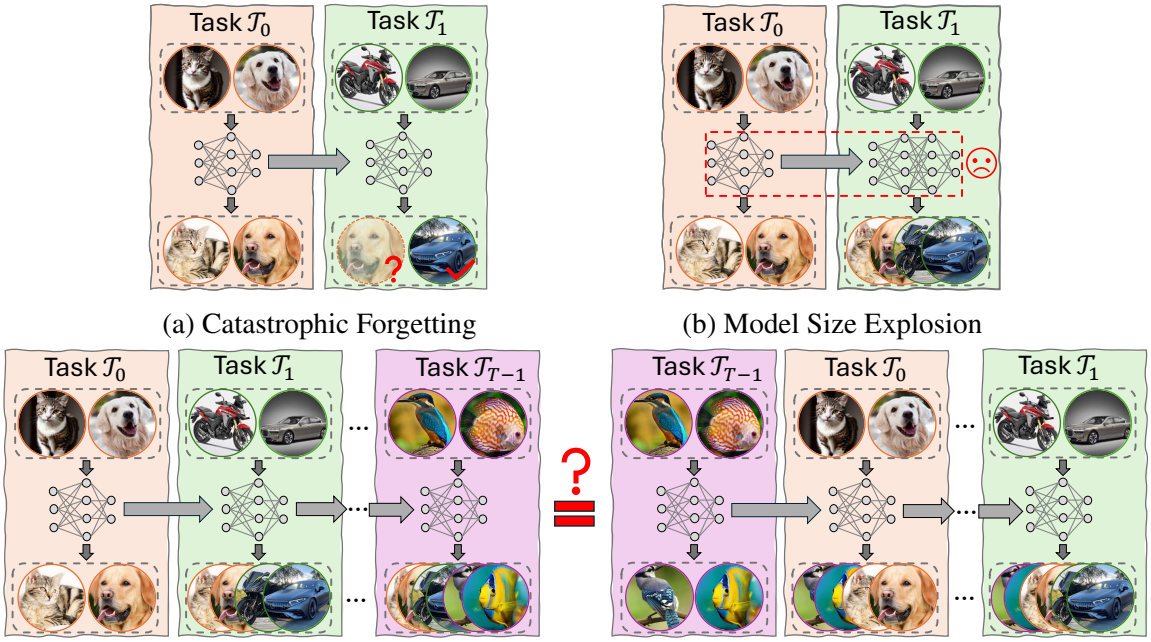
- *Domain Incremental Learning* (DIL): all the tasks $\{\mathcal{T}_0, \ldots, \mathcal{T}_{T-1}\}$ have the same class space (i.e., $\mathcal{Y}_0 = \cdots = \mathcal{Y}_{T-1}$) but follow different data distributions.

- *Task Incremental Learning* (TIL): the tasks have disjoint label spaces (i.e., $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset$ for $i \neq j$) as illustrated in Fig. 2.1, and the task identities are known during both training and testing procedures. Specifically, the model f maintains a set of *disjoint* classification heads $C_0, \ldots, C_{T-1}$ for each task, and during the training or evaluation process, the samples from a certain task $\mathcal{T}_t$ will be merely fed into the associated classification head C_t to obtain the classification probabilities.

- *Class Incremental Learning* (CIL): the tasks are also drawn from the disjoint class spaces as TIL, but the task identities are only accessed during training. That is, training samples from a certain task $\mathcal{T}_t$ are still fed into the associated classifier C_t during training; however, different from the TIL setting, a test sample does not contain the task identity information during the evaluation so that the model cannot directly process it with a specific classification head. In contrast, the test samples need to be processed by all the classification heads, and the predicted probabilities from all the classification heads will be used to determine the task identity and the class label. The CIL setting is more challenging compared to TIL, as inter-task confusion can exist during the evaluation.

In all of the above scenarios, the performances on all tasks currently seen (i.e., $\mathcal{T}_0, \ldots, \mathcal{T}_b$) need to be evaluated after learning a certain task $\mathcal{T}_b$ (with $0 \leq b \leq T-1$), as shown in Fig. 2.1.

This thesis focused mainly on the TIL setting, but our proposed method can be easily modified to adapt to other incremental learning scenarios.



(a) Catastrophic Forgetting

(b) Model Size Explosion

(c) Robustness under Different Task Orders

Figure 2.2: Challenges in continual learning

2.1.2 Challenges in Continual Learning

Since the knowledge from different tasks needs to be learned sequentially in continual learning, a common phenomenon is that the model usually loses the knowledge gained from the previous tasks after mastering the knowledge in new ones, i.e., *catastrophic forgetting* shown in Fig. 2.2a.

To overcome the forgetting issue, some studies proposed to maintain a small set of representative samples (i.e., *exemplars*) for each previous task and then replay them when learning new tasks, i.e., *rehearsal-based methods*. Specifically, after learning task $\mathcal{T}_b$, the exemplar set $\mathcal{E}$ is updated such that it contains some representative samples from all currently seen tasks, i.e., $\mathcal{E} \subset \cup_{t=0}^{b} \mathcal{D}_t$, then the learning process on the task $\mathcal{T}_{b+1}$ is conducted on not only the current task dataset $\mathcal{D}_{b+1}$ but the exemplars set $\mathcal{E}$:

$$\min_f \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_{b+1}\cup\mathcal{E}}\Big[\ell(f(\mathbf{x}),y)\Big] \tag{2.2}$$

where $\ell(\cdot,\cdot)$ is the loss function.

However, due to the potential risk of privacy leakage, the applications of this kind of method can be constrained in real-world scenarios. For example, when building an intelligent medical diagnosis system by continually learning from the sensitive data of different patients, maintaining such an exemplar set $\mathcal{E}$ with the private information of previous identities can be impractical or even not allowed when processing successive identities. Thus, it is crucial to investigate continual learning algorithms *without* accessing the data from previously learned tasks. In this thesis, we assume that the training data $\mathcal{D}_b$ becomes completely inaccessible after the training process on task $\mathcal{T}_b$ and the training data for the next task are only from $\mathcal{D}_{b+1}$, i.e., the more challenging *rehearsal-free* setting with $\mathcal{E} = \emptyset$.

Another popular strategy to address catastrophic forgetting is to isolate the knowledge from different tasks by adopting task-shared and task-private parameters in the model. This stream of approaches is usually termed as *expansion-based methods*. Specifically, denoting the task-shared parameters as $\mathbf{W}_s$ and the task-private parameters for t-th task $\mathcal{T}_t$ as $\mathbf{W}_{p_t}$, the learning process gradually expands the model by introducing private parameters w_{p_t} for newly coming tasks:

$$\overset{\text{learn } \mathcal{T}_0}{\longrightarrow} f_{\{\mathbf{W}_s\}\cup\{\mathbf{W}_{p_0}\}} \overset{\text{learn } \mathcal{T}_1}{\longrightarrow} f_{\{\mathbf{W}_s\}\cup\{\mathbf{W}_{p_0},\mathbf{W}_{p_1}\}} \longrightarrow \cdots \overset{\text{learn } \mathcal{T}_{T-1}}{\longrightarrow} f_{\{\mathbf{W}_s\}\cup\{\mathbf{W}_{p_0},\mathbf{W}_{p_1},\cdots,\mathbf{W}_{p_{T-1}}\}} \tag{2.3}$$

While this model expansion strategy can effectively diminish the knowledge forgetting on previous tasks, the size of private parameters within individual tasks $\{\mathbf{W}_{p_0}, \mathbf{W}_{p_1}, \ldots, \mathbf{W}_{p_{T-1}}\}$ will significantly increase as the task capacity grows, leading to the undesirable *model size explosion* issue (illustrated in Fig. 2.2b). Thus, it is crucial to achieve a good trade-off between performance and model increment. In this thesis, we adopted the form of residual learning and low-rank approximation.

Furthermore, some research [2] showed that the performance of each task can be incon-

sistent when the model receives the task sequence with different task orders, raising the concern of the *robustness on task orders* (as shown in Fig. 2.2c). For example, by denoting A_t^1, A_t^2, A_t^3 the performances of task $\mathcal{T}_t$ under three different task orders, the performance disparity $\delta_t = \max\{A_t^1, A_t^2, A_t^3\} - \min\{A_t^1, A_t^2, A_t^3\}$ reflects the performance fluctuation on task $\mathcal{T}_t$. Smaller δ_t indicates that the continual learning process is more robust. Then enhancing the task order robustness aims to achieve smaller δ_t for all tasks $t = 0, 1, \ldots, T-1$. This topic is under-explored in the context of continual learning, but it is crucial to investigate the robustness of continual learning algorithms from a novel perspective.

In this thesis, we tried to achieve a better trade-off among these challenging requirements.

2.1.3 Low-Rank Approximation of Matrices

The singular value decomposition (SVD) factorizes a rectangular matrix $\mathbf{B} \in \mathbb{R}^{J \times I}$ with three matrices: $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{J \times J}$, $\mathbf{V} \in \mathbb{R}^{I \times I}$, and $\mathbf{\Sigma} \in \mathbb{R}^{J \times I}$. Denote the rank of $\mathbf{B}$ as r (i.e., with $r \leq \min\{I, J\}$), then $\mathbf{B}$ can be further expressed as $\mathbf{B} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$, where $\sigma_i \in \text{diag}(\mathbf{\Sigma})$ is the singular values, $\mathbf{u}_i$ and $\mathbf{v}_i$ are respectively the left and right singular vectors. In this work, we take the property of k-rank approximation of $\mathbf{B}$ (with $k \leq r$) by leveraging the *Eckart–Young–Mirsky theorem* [13], which can be expressed with *top-k leading* singular values and singular vectors[2]:

$$\mathbf{B}^{(k)} = \mathbf{U}^{(k)}\mathbf{\Sigma}^{(k)}(\mathbf{V}^{(k)})^\top = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \tag{2.4}$$

with $k \leq r$, where $\mathbf{U}^{(k)}$, $\mathbf{\Sigma}^{(k)}$, $\mathbf{V}^{(k)}$ are the corresponding *leading* principal sub-matrices[3]. k is chosen to tolerate a certain approximation error under the Frobenius norm $\|\cdot\|_F$ (See

[2] In this thesis, we assume the eigenvalues are always sorted with descending order in such SVD decompositions, i.e., $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r$.

[3] With a slight abuse of notation, the notation $\mathbf{M}^{(k)}$ refers to the k-rank approximation of the matrix $\mathbf{M}$ according to the context: (1) $\mathbf{U}^{(k)}, \mathbf{\Sigma}^{(k)}, \mathbf{V}^{(k)}$ indicate the top-k leading submatrices; (2) $\mathbf{B}^{(k)}$ means the r-rank low-rank approximation for $\mathbf{B}$, then same for $\mathbf{W}^{(k)}$.

Appendix A for the proof):

$$\|\mathbf{B} - \mathbf{B}^{(k)}\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}. \tag{2.5}$$

2.2 Related Work

In this part, we mainly introduce existing studies in three typical technical streams of continual learning in the first three paragraphs. Then, in the last two paragraphs, we discuss the applications of low-rank factorization in continual learning and some relevant deep learning settings.

Regularization-based approaches This kind of method tried to diminish catastrophic forgetting by penalizing the parameter drift from the previous tasks using different regularizers [7–9]. Batch Ensemble [14] designed an ensemble weight generation method by the Hadamard product between a shared weight among all ensemble members and an ensemble member-specific rank-one matrix. [15] indicated that learning tasks in different low-rank vector subspaces orthogonal to each other can minimize task interferences. [16] applied a regularizer with decoupled prototype-based loss, which can improve the intra-class and inter-class structure significantly. Compared to this kind of approach, our method applied task-specific parameters and introduced an explicit weight transition process to leverage the knowledge from the previous tasks and overcome the forgetting issue.

Expansion-based methods Studies with model expansion utilize different subsets of model parameters for each task. [17] proposed to memorize the learned knowledge by freezing the base model and progressively expanding the new sub-model for new tasks. In [18], the reuse and expansion of networks were achieved dynamically by selective retraining, with splitting or duplicating components for newly coming tasks. Some work (*e.g.*, [19]) tried to determine the optimal network growth by neural architecture search. To

balance memory stability and learning plasticity, [20] adopted a distillation-based method under the class incremental scenario. To achieve the scalability and the robustness of task orders, Additive Parameter Decomposition (APD) [2] adopted sparse task-specific parameters for novel tasks in addition to the dense task-shared ones and performed hierarchical consolidation within similar task groups for the further knowledge sharing. [21] introduced Channel-Wise Linear Reprogramming (CLR) transformations on the output of each convolutional layer of the base model as the task-private parameters. However, this method relied on the prior knowledge from a disjoint dataset (e.g., ImageNet-1K). Winning Subnetworks (WSN) [1] jointly learned the shared network and binary masks for each task. To address the forgetting issue, only the model weights that had not been selected in the previous tasks were tendentiously updated during training. But this method still needs to store task-specific masks for the inference stage and relies on extra compression processes for mask encoding to achieve scalability.

Rehearsal-based approaches These methods leverage different types of replay buffers (*e.g.*, [15, 22–27]) to memorize a small episode of the previous tasks, which can be rehearsed when learning the novel ones to avoid forgetting. In [27], a dynamic prototype-guided memory replay module was incorporated with an online meta-learning framework to reduce memory occupation. [28] proposed to process both specific and generalized information by the interplay of three memory networks. To avoid the repeated inferences on previous tasks, Gradient Episodic Memory [29] and its variant [30] projected the new gradients into a feasible region that is determined by the gradients on previous task samples. Instead, Gradient Projection Memory (GPM) [5] chose to directly store the bases of previous gradient spaces to guide the direction of parameters update on new tasks. Compared to these approaches, our proposed method does not rely on extra storage for the data or gradient information of previous tasks, avoiding privacy leakage under some safety-sensitive scenarios.

Low-Rank Factorization for CL Low-rank factorization [31] has been widely studied in deep learning to decompose the parameters for model compression [32, 33] or data projection [34, 35]. In the context of CL, [12] considers the low-rank model factorization and the automatic rank selection per task for variational inference, which requires significant large rank increments per task to achieve high accuracy. GPM [5] applied the singular value decomposition on the representations and store them in the memory. [15] proposed to learn tasks by low-rank vector sub-spaces to avoid a joint vector space that may lead to interferences among tasks. The most similar work is Incremental Rank Updates (IRU) [3]. Compared to the decomposition in IRU, we adopted the low-rank approximation on the residual representation in the weights transitions. Furthermore, the rank selection in our work was dynamically and automatically determined according to Hessian-aware perturbations, rather than the manual rank increment in [3].

Low-Rank Adaptation for Foundation Models Some recent work also took advantage of the low-rank decomposition on adapting the *foundation models* (e.g., Large Language Model) pre-trained on large-scale datasets to the downstream tasks. Low-Rank Adapter (LoRA) [36] was the first and representative work. Different from the full fine-tuning that needs to update all the parameters in the model, this method proposed to freeze the pre-trained parameters and plug in a parallel residual branch for each layer (i.e., an attention block in the Transformer-based model [37]) when adapting to the downstream tasks. This residual branch consists of a scale-down and a scale-up operations, forming a low-rank bottleneck and aiming to map the feature into a low-dimensional space and then back to the original feature space. As one of the *parameter-efficient fine-tuning* strategies, LoRA and its variants (e.g., [38–40]) adapt the pre-trained models onto the downstream tasks without introducing numerous parameters and efficiently transfer the knowledge. Compared to these methods, our proposed method mainly focused on the continual learning setting where there exists a long sequence of tasks and overcoming the forgetting issue is still

challenging. In addition, the rank numbers within each layer of the deep model were automatically determined through the Hessian information in our proposed method, rather than being manually set like the above-mentioned low-rank adapters.

Chapter 3

3 Hessian-aware Low-rank Perturbation for Continual Learning

[1] Our framework leverages the low-rank approximation of neural network weights. In the

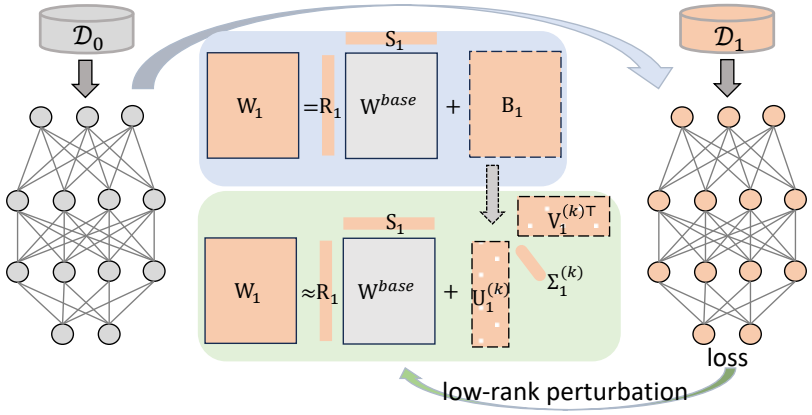


Figure 3.1: Overview of the proposed method: an example from $\mathcal{T}_0$ to $\mathcal{T}_1$

following parts, we present the methodology for handling the fully connected layers and the convolutional layers. The analysis herein can be applied to any layer of the model. Without loss of generality, we omit the layer index l in this section. For simplicity, we illustrate the learning process using tasks $\mathcal{T}_0$ and $\mathcal{T}_1$ in Sections 3.1 and 3.2, which can be applied to the successive new tasks as shown in Section 3.6.

3.1 Parameter Transition between Tasks: Linear Layers

We first consider linear layers of neural networks. We begin by learning task $\mathcal{T}_0$ without any constraints on the model parameters. Specifically, we train the model by minimizing

[1] A version of this chapter has been published in [41].

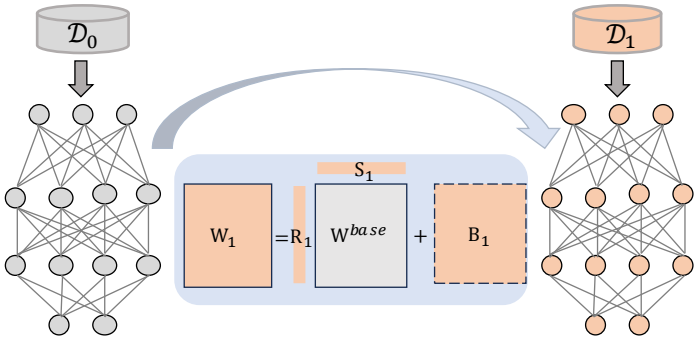


Figure 3.2: Parameter transition from $\mathcal{T}_0$ to $\mathcal{T}_1$

the empirical risk to get the base weights:

$$\mathbf{W}^{\text{base}} = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}; \mathcal{D}_0) \tag{3.1}$$

where $\mathbf{W}^{\text{base}} \in \mathbb{R}^{J \times I}$, J is the output dimension, and I is the input dimension of the layer, $\mathcal{L}$ is the training loss function (e.g., cross-entropy loss).

Then, when the task $\mathcal{T}_1$ comes to the learner, we can train the model fully on $\mathcal{D}_1$ and get the updated weights $\mathbf{W}_1 \in \mathbb{R}^{J \times I}$. However, undesired parameter drifts can occur when learning new knowledge from $\mathcal{T}_1$ and lead to worse performance on the previous task $\mathcal{T}_0$. Previous work [7] applied a L_2 regularization to ensure that the weights learned on the new model will not be too far from those of the previous tasks. Although this kind of method only expands the base model with limited parameters, it can lead to worse performance on both $\mathcal{T}_0$ and $\mathcal{T}_1$.

To pursue a better trade-off between the model size increment and overall performance on both $\mathcal{T}_0$ and $\mathcal{T}_1$, we assume the unconstrained trained parameters $\mathbf{W}_1$ on $\mathcal{T}_1$ can be transformed from $\mathbf{W}^{\text{base}}$ obtained from $\mathcal{T}_0$ by the *low-rank weight perturbation* (LRWP, as illustrated in Figure 3.1),

$$\mathbf{W}_1 = \mathbf{R}_1 \mathbf{W}^{\text{base}} \mathbf{S}_1 + \mathbf{B}_1 \tag{3.2}$$

where $\mathbf{B}_1 \in \mathbb{R}^{J \times I}$ is a residual low-rank matrix, $\mathbf{R}_1 = \text{Diag}(r_1, \ldots, r_J)$, and $\mathbf{S}_1 = \text{Diag}(s_1, \ldots, s_I)$

are scaling parameters[2] for $\mathcal{T}_1$, as illustrated in Fig. 3.2. These introduced parameters will be determined through Eq. 3.3 in the following part.

The form of the proposed low-rank decomposition in Eq. 3.2 has differences with the additive parameter decomposition proposed in [2]. In fact, the task-adaptive bias term $\mathbf{B}_1$ conforms to a low-rank matrix, which reduces both the parameter storage and the computational overhead. Moreover, the task-adaptive mask terms $\mathbf{R}_1$ and $\mathbf{S}_1$ include both row-wise and column-wise scaling parameters instead of only column-wise parameters like [2] to allow smaller and possibly sparser discrepancy: $\mathbf{B}_1 = \mathbf{W}_1 - \mathbf{R}_1\mathbf{W}^{\text{base}}\mathbf{S}_1$.

In addition, regarding the parameter estimation, we can substitute $\mathbf{B}_1$ with its k-rank approximation $\mathbf{B}_1^{(k)} = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ in Eq. 3.2 and directly minimize the empirical risk on task $\mathcal{T}_1$ to get the parameter estimation through stochastic gradient descent with the random initialization of $\{\mathbf{u}_i\}_{i=1}^{k}$ and $\{\mathbf{v}_i\}_{i=1}^{k}$. However, in practice, we found it challenging to learn a useful decomposition in this way (*i.e.*, the estimations converge to a local optimum quickly) and the empirical risk minimization does not benefit from the low-rank decomposition. Due to the homogeneity among the one-rank components $\{\mathbf{u}_i\mathbf{v}_i^\top\}_{i=1}^{k}$, random initialization cannot sufficiently distinguish them and hence make the gradient descent ineffective.

To address the aforementioned ineffective training problem, we first train the model fully on task $\mathcal{T}_1$ for a few epochs (*e.g.*, one or two) to get rough parameter estimations $\mathbf{W}_1^{\text{free}} \in \mathbb{R}^{J\times I}$ of the new task, indicating free-trained weights without any constraints. Then, a good warm-up initialized values for Eq. 3.2 can then be obtained by solving the following least squared error (LSE) minimization objective,

$$\arg\min_{\mathbf{R},\mathbf{S},\mathbf{B}} \|\mathbf{W}_1^{\text{free}} - \mathbf{R}\mathbf{W}^{\text{base}}\mathbf{S} - \mathbf{B}\|_F^2 \tag{3.3}$$

As $\mathbf{R}, \mathbf{S}, \mathbf{B}$ can correlate with each other, minimizing them simultaneously can be difficult.

[2] We will show later that $\mathbf{B}$ can be further approximated by $\mathbf{U}^{(k)}, \mathbf{\Sigma}^{(k)}, \mathbf{V}^{(k)}$ with low-rank approximation, so we can write the task-private weights for task t as either $\{\mathbf{R}_t, \mathbf{S}_t, \mathbf{B}_t^{(k)}\}$ or $\{\mathbf{R}_t, \mathbf{S}_t, \mathbf{U}_t^{(k)}, \mathbf{\Sigma}_t^{(k)}, \mathbf{V}_t^{(k)}\}$.

Thus, we solve the problem in Eq. 3.3 alternately. Specifically, we first fix $\mathbf{S}, \mathbf{B}$ and solve $\mathbf{R}$, and then solve $\mathbf{S}$, etc.

$$\begin{aligned}
\mathbf{R}_1^{\text{free}} &= \arg\min_{\mathbf{R}} \|\mathbf{W}_1^{\text{free}} - \mathbf{R}\mathbf{W}^{\text{base}}\|_F^2 \\
\mathbf{S}_1^{\text{free}} &= \arg\min_{\mathbf{S}} \|\mathbf{W}_1^{\text{free}} - \mathbf{R}_1^{\text{free}}\mathbf{W}^{\text{base}}\mathbf{S}\|_F^2 \\
\mathbf{B}_1^{\text{free}} &= \mathbf{W}_1^{\text{free}} - \mathbf{R}_1^{\text{free}}\mathbf{W}^{\text{base}}\mathbf{S}_1^{\text{free}}
\end{aligned} \tag{3.4}$$

Remark Obviously, $\|\mathbf{W}^{\text{free}} - \mathbf{R}\mathbf{W}^{\text{base}}\|_F^2 = \sum_{i=1}^{I}\sum_{j=1}^{J}(w_{ji}^{\text{free}} - r_j w_{ji}^{\text{base}})^2$, where w_{ji}^{free} and w_{ji}^{base} are the elements of j-th row and i-th column of $\mathbf{W}^{\text{free}}$ and $\mathbf{W}^{\text{base}}$, respectively. By taking the derivative of the above expression w.r.t. r_j and let it equal 0, we can obtain:

$$r_j^{\text{free}} = \frac{\sum_{i=1}^{I} w_{ji}^{\text{free}} w_{ji}^{\text{base}}}{\sum_{i=1}^{I}(w_{ji}^{\text{base}})^2} \tag{3.5}$$

After obtaining r_j^{free}, we can similarly have $\|\mathbf{W}^{\text{free}} - \mathbf{R}^{\text{free}}\mathbf{W}^{\text{base}}\mathbf{S}\|_F^2 = \sum_{i=1}^{I}\sum_{j=1}^{J}(w_{ji}^{\text{free}} - r_j^{\text{free}} w_{ji}^{\text{base}} s_i)^2$. By taking the derivative w.r.t. s_i and let it equal 0, we can obtain:

$$s_i^{\text{free}} = \frac{\sum_{j=1}^{J} r_j^{\text{free}} w_{ji}^{\text{free}} w_{ji}^{\text{base}}}{\sum_{j=1}^{J}(r_j^{\text{free}} w_{ji}^{\text{base}})^2} \tag{3.6}$$

Then, we can calculate $\mathbf{B}^{\text{free}}$ by the third equation using the obtained $\mathbf{R}^{\text{free}}$ and $\mathbf{S}^{\text{free}}$. □

3.2 Parameter Transition between Tasks: Convolutional Layers

In addition, we can decompose the weights of a convolutional layer in a similar way. Suppose that the size of the convolutional kernel is $d \times d$. The base weights of the convolution layer for task $\mathcal{T}_0$ would be a tensor $\mathbf{W}_{\text{conv}}^{\text{base}} \in \mathbb{R}^{d\times d\times J\times I}$. Similar to Eq. 3.2, a low-rank weight

perturbation for transforming $\mathbf{W}_{\text{conv}}^{\text{base}}$ to $\mathbf{W}_{\text{conv},1} \in \mathbb{R}^{d\times d\times J\times I}$ is,

$$\mathbf{W}_{\text{conv},1} = \mathbf{R}_{\text{conv},1} \otimes \mathbf{W}_{\text{conv}}^{\text{base}} \otimes \mathbf{S}_{\text{conv},1} \oplus \mathbf{B}_{\text{conv},1} \tag{3.7}$$

where $\mathbf{R}_{\text{conv},1} \in \mathbb{R}^{1\times 1\times J\times 1}$, $\mathbf{S}_{\text{conv},1} \in \mathbb{R}^{1\times 1\times 1\times I}$ and $\mathbf{B}_{\text{conv},1} \in \mathbb{R}^{1\times 1\times J\times I}$ is sparse low-rank tensor (matrix), $\otimes$ and $\oplus$ are element-wise tensor multiplication and summation operators that will automatically expand tensors to be of equal sizes, following the broadcasting semantics of some popular scientific computation package like Numpy [42] or PyTorch [43]. Thus, the number of parameters added through this kind of decomposition is still $O(I + J)$.

To get the estimations of the introduced parameters, we can solve a similar LSE problem as in Eq. 3.3 to obtain initial estimates $\mathbf{R}_{\text{conv},1}^{\text{free}}$ and $\mathbf{S}_{\text{conv},1}^{\text{free}}$. Then we take the average of the first two dimensions to transform the discrepancy tensor $(\mathbf{W}_{\text{conv},1}^{\text{free}} - \mathbf{R}_{\text{conv},1}^{\text{free}} \otimes \mathbf{W}_{\text{conv}}^{\text{base}} \otimes \mathbf{S}_{\text{conv},1}^{\text{free}})$ to be a $1\times 1\times J\times I$ tensor, which can then be applied a similar decomposition with the linear layers to obtain the low-rank estimates of $\mathbf{B}_{\text{conv},1}^{\text{free}}$, as described in the following section.

3.3 Model Increment Control via Low Rank Approximation

Furthermore, with the SVD solver indicated by $\texttt{SVD}(\cdot)$, the values of the low-rank decomposition for $\mathbf{B}^{\text{free}}$ can be obtained:

$$\mathbf{U}_1^{\text{free}}, \mathbf{\Sigma}_1^{\text{free}}, \mathbf{V}_1^{\text{free}} \leftarrow \texttt{SVD}(\mathbf{B}_1^{\text{free}}) \tag{3.8}$$

A k-rank approximation $\mathbf{U}_1^{(k)\text{free}}$, $\mathbf{\Sigma}_1^{(k)\text{free}}$, $\mathbf{V}_1^{(k)\text{free}}$ can be obtained by retaining their corresponding leading principal submatrix of order $k \leq r$. So we can obtain a k-rank approximation to $\mathbf{W}_1^{\text{free}}$ by

$$\mathbf{W}_1^{\text{free}} \approx \mathbf{W}_1^{(k)\text{free}} = \mathbf{R}_1^{\text{free}} \mathbf{W}^{\text{base}} \mathbf{S}_1^{\text{free}} + \mathbf{B}_1^{(k)\text{free}}, \tag{3.9}$$

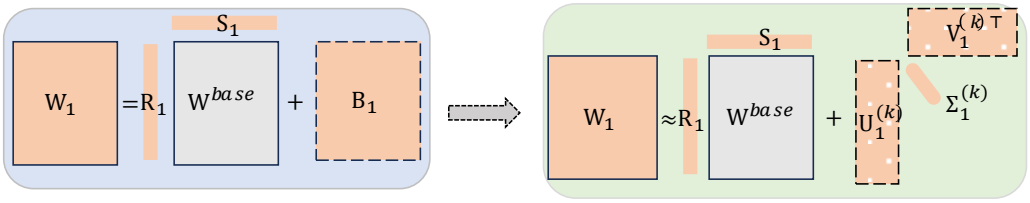


Figure 3.3: Low-rank approximation on weights

with

$$\mathbf{B}_1^{(k)\text{free}} = \mathbf{U}_1^{(k)\text{free}} \mathbf{\Sigma}_1^{(k)\text{free}} (\mathbf{V}_1^{(k)\text{free}})^\top \tag{3.10}$$

as illustrated in Fig. 3.3

Remark Note that the number of parameters in the original weights $\mathbf{W}^{\text{free}} \in \mathbb{R}^{J \times I}$ is $\text{sizeof}\{\mathbf{W}^{\text{free}}\} = JI$. After this approximation, we only need to store the weights $\mathbf{R}^{\text{free}}$, $\mathbf{S}^{\text{free}}$, $\mathbf{U}^{(k)\text{free}}$, $\mathbf{\Sigma}^{(k)\text{free}}$, $\mathbf{V}^{(k)\text{free}}$ for a new task, whose parameter number is

$$\text{sizeof}\{\mathbf{R}^{\text{free}}, \mathbf{S}^{\text{free}}, \mathbf{U}^{(k)\text{free}}, \mathbf{\Sigma}^{(k)\text{free}}, \mathbf{V}^{(k)\text{free}}\} = J + I + kJ + k + kI = (J + I)(k + 1) + k \tag{3.11}$$

Thus, the incremental ratio is $\rho = \frac{(J+I)(k+1)+k}{JI} \ll 1$ in practice. □

Finally, we can initialize the values in Eq. 3.2 with $\mathbf{R}_1^{\text{free}}$, $\mathbf{S}_1^{\text{free}}$, $\mathbf{U}_1^{(k)\text{free}}$, $\mathbf{\Sigma}_1^{(k)\text{free}}$ and $\mathbf{V}_1^{(k)\text{free}}$, and then fine-tune their estimates by minimizing the empirical risk on task $\mathcal{T}_1$ to achieve better performance. The proposed training technique not only enables well-behaved estimations of the low-rank components but sheds light on how to select approximation ranks of different layers to achieve an optimal trade-off between model performance and parameter size, as discussed in Section 3.4.

3.4 Rank Selection via Hessian Aware Perturbation

Sections 3.1 and 3.2 present how the low-rank approximation can be used to transfer knowledge and reduce the number of parameters for a single layer across the tasks in continual

learning. However, how to select the preserved rank for each layer remains unsolved.

Generally, more ranks can benefit the approximation process introduced in the previous section. However, keeping more ranks will naturally lead to a higher model increment. Thus, the preserved rank number for each layer within the model should be selected properly. In our work, we tackle this problem by measuring how the empirical risk $\mathcal{L}(\mathbf{W})$ is influenced by the introduced low-rank parameters across different layers so that we can assign a larger rank to a layer that contributes more to the risk.

Inspired by the previous studies [44, 45] about the relationship between Hessian and quantization errors, we establish the following Theorem 1. The full proof is presented in Appendix B.

Theorem 1. *Assume that a neural network of L layers with vectorized weights* $(\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star)$ *that have converged to local optima, such that the first and second order optimally conditions are satisfied,* i.e.*, the gradient is zero, and the Hessian is positive semi-definite. Suppose a perturbation* $\Delta\boldsymbol{\omega}_1^\star$ *applied to the first layer weights, then we have the loss change*

$$\left|\mathcal{L}(\boldsymbol{\omega}_1^\star - \Delta\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star) - \mathcal{L}(\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star)\right| \leq \frac{1}{2}\|\mathbf{H}_1\|_F \cdot \|\Delta\boldsymbol{\omega}_1^\star\|_F^2 + o\left(\|\Delta\boldsymbol{\omega}_1^\star\|_F^2\right), \tag{3.12}$$

where $\mathbf{H}_1 = \nabla^2\mathcal{L}(\boldsymbol{\omega}_1^\star)$ *is the Hessian matrix at only the variables of the first layer weights.*

Remark Theorem 1 demonstrated the relationship between the perturbation $\Delta w_1^\star$ on weights and the effect on the loss objective $\Delta\mathcal{L}$. Specifically, when a weight perturbation $\Delta w_1^\star$ is applied to the related weight matrix $w_1^\star$, the perturbation introduced on the loss function is upper bounded mainly by the product of the Frobenius norms of Hessian matrix (i.e., $\|\mathbf{H}_1\|_F$) and weight perturbation (i.e., $\|\Delta\boldsymbol{\omega}_1^\star\|_F^2$). It further inspires us to follow this rule to select the proper ranks by considering the low-rank approximation in the previous section as a perturbation to the model weights. □

In our low-rank perturbation setting, we assume that $\mathbf{W}_1^{\text{free}}$ by warm-up training is a local

optimum. By Theorem 1, we consider the difference between $\mathbf{W}_1^{\text{free}}$ and its k-rank approximation $\mathbf{W}_1^{(k)\text{free}}$ as a perturbation $\Delta\mathbf{W}_1^{\text{free}}$ for the weights. Then the amount of perturbation can be computed with the low-rank approximation error:

$$\|\Delta\mathbf{W}_1^{\text{free}}\|_F = \|\mathbf{W}_1^{\text{free}} - \mathbf{W}_1^{(k)\text{free}}\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2} \tag{3.13}$$

where $\{\sigma_i\}_{i=1}^r$ are the singular values of $\mathbf{W}_1^{\text{free}}$ and r is the matrix rank of $\mathbf{W}_1^{\text{free}}$.

Thus, according to Theorem 1, the influence on the loss introduced by this low-rank weight approximation is given by

$$\left|\mathcal{L}(\mathbf{W}_1^{(k)\text{free}}) - \mathcal{L}(\mathbf{W}_1^{\text{free}})\right| \leq \frac{1}{2}\|\mathbf{H}_1\|_F \cdot \left(\sum_{i=k+1}^{r} \sigma_i^2\right) + o\left(\sum_{i=k+1}^{r} \sigma_i^2\right). \tag{3.14}$$

where the Hessian matrix $\mathbf{H}_1$ can be approximated by the empirical negative Fisher information [46], *i.e.*, the outer product of the gradient vector for the layer weights. So $\|\mathbf{H}_1\|_F$ can be approximated by $\|\mathbf{g}_1\|_2^2$, where $\mathbf{g}_1 = \frac{\partial\mathcal{L}}{\partial\mathbf{W}_1}|_{\mathbf{W}_1=\mathbf{W}_1^{\text{free}}}$. Finally, we can quantitatively measure the contribution of the loss of adding a marginal rank k for a particular layer l by

$$\|\mathbf{g}_l\|_2^2 \sigma_{l,k}^2 \tag{3.15}$$

where $\mathbf{g}_l$ is the gradient for the layer-l weights and $\sigma_{l,k}$ is the k-th singular value of the free-trained layer-l weights, and sort them by the descending order of importance.

For a given loss approximation rate α (*e.g.*, 0.9), we can determine the rank k_l (with $k_l \leq r_l$

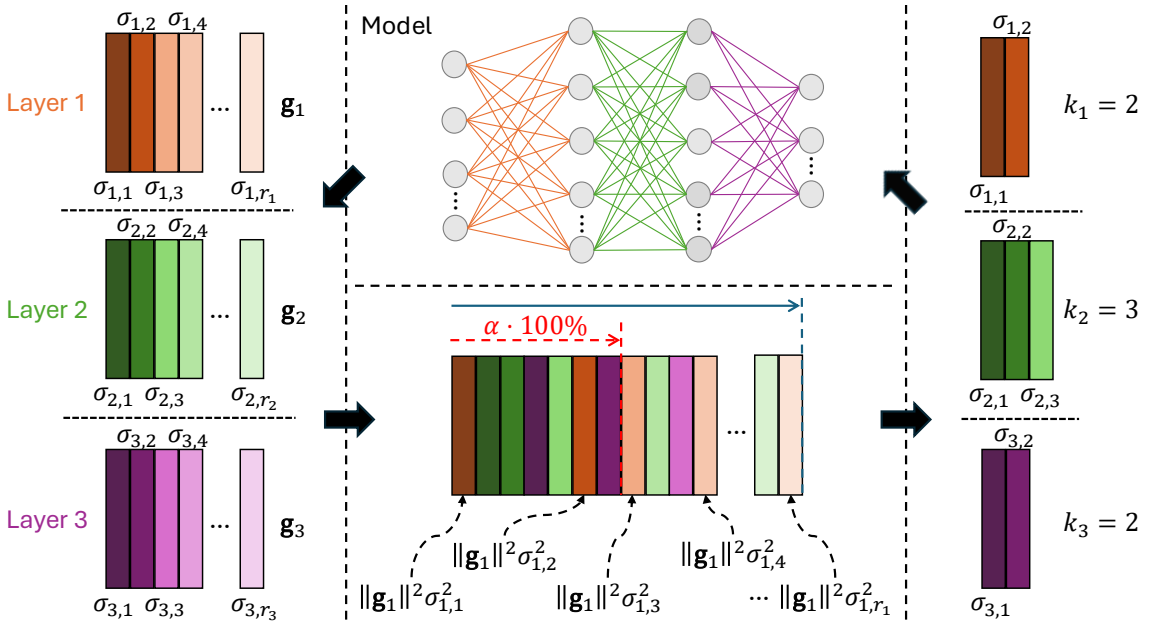


Figure 3.4: Rank selection process (an example for $L = 3$)

where r_l is the total rank of the layer l) for each layer $l = 1, ..., L$ by solving

$$\begin{aligned} &\min_{k_1,\ldots,k_L} \quad \sum_{l=1}^{L}\sum_{i=1}^{k_l} \|\mathbf{g}_l\|_2^2 \sigma_{l,i}^2 \\ &\text{s.t.} \sum_{l=1}^{L}\sum_{i=1}^{k_l} \|\mathbf{g}_l\|_2^2 \sigma_{l,i}^2 \geq \alpha \cdot \left(\sum_{l=1}^{L}\sum_{i=1}^{r_l} \|\mathbf{g}_l\|_2^2 \sigma_{l,i}^2 \right) \end{aligned} \tag{3.16}$$

Specifically, Fig. 3.4 provides an example of Eq. 3.16 for the rank selection process within a three-layer neural network. Firstly, the gradient information $\mathbf{g}_l$ and the rank values $\sigma_{l,1}, \ldots, \sigma_{l,r_l}$ are obtained as described in the previous sections. Secondly, the contributions of each rank within each layer of this three-layer model can be measured by the gradient-weighted singular values $\|\mathbf{g}_l\|^2\sigma_{l,k}^2$ and they can be sorted with the descending order. Then, given a predefined approximation rate α, the ranks are truncated to achieve such an explanation ratio and the number of preserved ranks k_l for each layer can be accordingly determined (i.e., $k_1 = 2, k_2 = 3, k_3 = 2$ for three layers in this example, respectively.) Finally, the parameters of the preserved ranks are adopted to reinitialize the model parameters for further fine-tuning.

Remark Eq. 3.16 enables a dynamic scheme for the trade-off between approximation precision and computational efficiency. For a given approximation rate, the model can automatically select the ranks for all the layers in the model. □

3.5 Regularization and Pruning on Parameters

The proposed low-rank perturbation method introduced extra parameters compared to a single-task model. For these parameters, we can further apply regularizations to avoid overfitting. To control the model growth, we can further prune the introduced parameters to improve memory efficiency.

Firstly, by following [2], we can add regularizations on $\mathbf{U}^{(k)\text{free}}$, $\mathbf{V}^{(k)\text{free}}$, $\mathbf{R}^{\text{free}}$,$\mathbf{S}^{\text{free}}$ since second task $\mathcal{T}_1$ to enhance the sparsity of the task-private parameters (see Appendix C for more discussion for this fine-tuning objective):

$$\mathcal{L}_{\text{reg}}(\mathbf{W}) = \sum_{l=1}^{L}\Big[\lambda_0\Big(\|\mathbf{U}_{t,l}^{(k_l)\text{free}}\| + \|\mathbf{V}_{t,l}^{(k_l)\text{free}}\|\Big) + \lambda_1\Big(\|\mathbf{R}_{t,l}^{\text{free}}\|_2^2 + \|\mathbf{S}_{t,l}^{\text{free}}\|_2^2 + \|\mathbf{U}_{t,l}^{(k_l)\text{free}}\|_2^2 + \|\mathbf{V}_{t,l}^{(k_l)\text{free}}\|_2^2\Big)\Big] \tag{3.17}$$

where λ_0, λ_1 are balancing coefficients, and the subscripts (t, l) indicate the relevant weights for layer l in task t.

Thus, the total optimization objective for the task $\mathcal{T}_t$ with $t \geq 1$ with the regularization becomes:

$$\min_{\mathbf{W}}\Big[\mathcal{L}(\mathbf{W}; \mathcal{D}_t) + \mathcal{L}_{\text{reg}}(\mathbf{W})\Big] \tag{3.18}$$

Secondly, we can also prune the extra parameters by setting zero values for elements whose absolute values are lower than a certain threshold. The threshold can be selected in the following three ways:

(1) Pruning via absolute value: a fixed tiny positive value (*e.g.*, 10^{-5}) is set as the thresh-

Algorithm 1 Hessian Aware Low-Rank Perturbation (HALRP) for Continual Learning

Require: Task data $\{\mathcal{D}_t\}_{t=0}^{T-1}$; total epochs for one task n; rank estimation epochs n_r; parameter increments limitation ratio p, approximation rate α.

Ensure: Base weights $\mathbf{W}^{\text{base}}$ and $\{\mathbf{W}_t^\star\}_{t=1}^{T-1}$ for each task.

1: Obtain $\mathbf{W}^{\text{base}} = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}; \mathcal{D}_0)$ on task $\mathcal{T}_0$.
2: **for** $t = 1, \cdots, T-1$ **do**
3: Warm-up pre-training on task $\mathcal{T}_t$ for n_w epochs: $\mathbf{W}_t^{\text{free}} = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}; \mathcal{D}_t)$.
4: Low-rank decomposition for all layers via Eq. 3.2 or Eq. 3.7: $\mathbf{W}_t^{\text{free}} = \mathbf{R}_t^{\text{free}} \mathbf{W}^{\text{base}} \mathbf{S}_t^{\text{free}} + \mathbf{B}_t^{\text{free}}$.
5: Apply $\mathbf{U}_t^{\text{free}}, \mathbf{\Sigma}_t^{\text{free}}, \mathbf{V}_t^{\text{free}} \leftarrow \texttt{SVD}(\mathbf{B}_t^{\text{free}})$.
6: Select the ranks k_l for each layer l through Eq. 3.16.
7: Re-initialize the task $\mathcal{T}_t$ parameters with Eq. 3.9.
8: Fine-tuning on $\mathcal{T}_t$ for $(n - n_w)$ epochs with:

$$\mathbf{W}_t^\star = \arg\min_{\mathbf{W}} \left[\mathcal{L}(\mathbf{W}; \mathcal{D}_t) + \mathcal{L}_{\text{reg}}(\mathbf{W}) \right] \tag{3.19}$$

9: If the size of the introduced parameters is larger than a threshold p, apply the pruning method in Section 3.5.
10: **end for**
11: **return** $\mathbf{W}^{\text{base}}$ and $\{\mathbf{W}_t^\star\}_{t=1}^{T-1}$.

old and parameters less than this threshold will be set as zeros;

(2) Pruning via relative percentile: to control the ratio of increased parameter size over a single-task model size under γ (e.g., 40%), the pruning threshold is selected as the $(1-\gamma)$-percentile of the low-rank parameters among all layers of all tasks.

(3) Pruning via mixing absolute value and relative percentile: we set a threshold as the maximum of the thresholds obtained from the above two methods to prune using relative percentiles.

3.6 Summary of Proposed Algorithm

In the previous sections, we described the model update from $\mathcal{T}_0$ to $\mathcal{T}_1$ with an illustrative example. The overall description of our proposed HALRP[3] is shown in Algorithm 1.

[3] Our source code is at: https://github.com/lijiaqi/HALRP

At the start, the learner was trivially trained on the first task $\mathcal{T}_0$ to obtain $\mathbf{W}^{\text{base}}$. As for each incoming task $\mathcal{T}_t$ with $t = 1, ..., T-1$, we first train the model without any constraints for n_w epochs to get a rough initialization $\mathbf{W}_t^{\text{free}}$. Secondly, we apply the low-rank decomposition on all layers with Eq. 3.2 (for linear layers) or Eq. 3.7 (for convolutional layers) by solving a least square error minimization problem described in Eq. 3.4. Then, we further apply the singular value decomposition for the residual matrix $\mathbf{B}_t^{\text{free}}$. With this decomposition, we can measure the Hessian-aware perturbations and the rank k_l for each layer l through Eq. 3.16. After the rank selection, we can re-initialize the model parameters with the approximated weights $\mathbf{W}_t^{(k)\text{free}} \approx \mathbf{W}_t^{\text{free}}$ and then fine-tune the model for the remaining $n - n_w$ epochs to obtain the optimal weights for $\mathcal{T}_t$. As for the inference stage, the base weights and the task-specific parameters can be adopted to make predictions for each task.

Chapter 4

4 Experimental Results

We first compare the accuracy over several recent baselines with standard CL protocol. Then, we studied the task order robustness, forgetting, memory cost, training time efficiency, and the ablation study to show the effectiveness further. We briefly describe the experimental setting herein while delegating more details in Appendix E.

4.1 Experimental Settings

Datasets: We evaluate the algorithm on the following datasets:

- **CIFAR100-Split**: split the classes into 10 groups; each group is a 10-way classification task.
- **CIFAR100-SuperClass**: it consists of images from 20 superclasses of the CIFAR-100 dataset; each superclass contains five different class and forms a 5-way classification task.
- **Permuted MNIST (P-MNIST)**: obtained from the MNIST dataset [47] by random permutations of the original MNIST pixels. We follow [48] to create 10 sequential tasks using different permutations, and each task has 10 classes.
- **Five-dataset**: It uses a sequence of 5 different benchmarks including CIFAR10 [49], MNIST [47], notMNIST [50], FashionMNIST [51] and SVHN [52]. Each benchmark contains 10 classes.
- **Omniglot Rotation** [53]: 100 12-way classification tasks. The rotated images in 90°, 180°, and 270° are generated by following [2].
- **TinyImageNet**: a variant of ImageNet [54] dataset containing 200 classes. Here,

Dataset	#Tasks	#Classes/Task	#Train samples/Task	#Test samples/Task
CIFAR100-Split	10	10	5000	1000
CIFAR100-Superclass	20	5	2500	500
PMNIST	10	10	60000	10000
Five-dataset	5	10	50000~73257	10000~26032
OmniglotRotation	100	12	720	240
TinyImageNet (20-split)	20	10	5000	500
TinyImageNet (40-split)	40	5	2500	250

Table 4.1: Statistics of datasets adopted in the experiments.

we adopted two settings, one with 20 10-way classification tasks (**TiyImageNet 20-split**) and another more challenging setting with 40 5-way classification tasks (**TinyImageNet 40-split**).

The detailed statistics of the adopted datasets are summarized in Table 4.1.

Baselines: We compared the following baselines by the publicly released code or our re-implementation:

- **STL**: single-task learning with individual models for each task.
- **MTL**: multi-task learning with a single for all the tasks simultaneously [55].
- **EWC**: *Elastic Weight Consolidation* method proposed by [7].
- **L2**: the model is trained with L_2-regularizer [7] $\lambda \cdot \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\|_2^2$ between the current model and the previous one.
- **BN**: The *Batch Normalization* method [56].
- **BE**: The *Batch Ensemble* method proposed by [14].
- **APD**: The *Additive Parameter Decomposition* method [2]. Each layer of the target network was decomposed into task-shared and task-specific parameters with mask vectors.
- **APDfix**: We modify the APD method by fixing the model parameters while only learning the mask vector when a new task comes to the learner.

- **IBPWF**: Determine the model expansion with non-parametric Bayes and weights factorization [12].
- **GPM** [5]: A replay-based method by orthogonal gradient descent.
- **WSN** [1]: Introduce learnable weight scores to generate task-specific binary masks for optimal subnetwork selection.
- **BMKP** [57]: A bilevel memory framework for knowledge projection: a working memory to ensure plasticity and a long-term memory to guarantee stability.
- **CLR** [21]: An *expansion-based* method by applying Channel-Wise Linear Reprogramming transformations on each convolutional layer in the base model. This method originally relies on a model pre-trained on an extra dataset (i.e., ImageNet-1K) disjoint with the above task datasets, rather than the *train-from-scratch* manner adopted by other baselines. To make a fair comparison, we pre-trained the base model on the first task of the above task datasets and applied it to all tasks.
- **PRD** [58]: Prototype-sample relation distillation with supervised contrastive learning.
- **IRU** [3]: Furthermore, we also compared our methods with **IRU** [3], a method also based on the low-rank decomposition. However, due to the code limitation[1] of **IRU** [3], we further implemented our method under the dataset protocol and model architecture setting in [3] and compared our performance with the results reported in [3]. For the implementation of all the methods, we applied the same hyperparameters (e.g., batch size, training epochs, regularization coefficient) to realize fair comparisons. See Appendix E.3 for more details.

Model Architecture: For CIFAR100-Split, CIFAR100-SuperClass, and P-MNIST datasets,

[1] IRU authors only released the implementation on the multi-layer perception and did not release the code for convolutional neural networks. See https://github.com/CSIPlab/task-increment-rank-update for more details.

we adopted LeNet as the base model. As for Five-dataset and TinyImageNet datasets, we evaluated with AlexNet and reduced ResNet18 networks wehre the latter has reduced filters compared to standard ResNet18 (see Appendix E.2). And we followed [1] to use an extended LeNet model on the Omniglot-Rotation dataset.

Evaluation Metrics: We mainly adopted the average of the accuracies of the final model on all tasks (we will call "accuracy" or "Acc." later) for the empirical comparisons. Denote $A_{t,i}$ as the accuracy on task i after training on task t (with $i \leq t$). The final average accuracy (Acc.) can be defined as:

$$\text{Acc} \triangleq \frac{1}{T}\sum_{i=0}^{T-1} A_{T-1,i} \tag{4.1}$$

As for the forgetting statistics, we applied backward transfer (BWT) [5] as a quantitive measure:

$$\text{BWT} \triangleq \frac{1}{T}\sum_{i=0}^{T-1}(A_{i,i} - A_{T-1,i}) \tag{4.2}$$

Especially, we also compared the order-robustness of different methods by calculating the **Order-normalized Performance Disparity** that will be introduced later. For each single experiment (e.g., each task order), we repeat with five random seeds to compute the average and standard error.

4.2 Empirical Accuracy

We provide the average accuracies on the six benchmarks in Table 4.2, Table 4.3 and Table 4.4. From these numerical results, we can conclude our method achieved state-of-the-art performance compared to the baseline methods.

According to the evaluations on CIFAR100-Split and CIFAR100-SuperClass in Table 4.2, we can observe that our proposed HALRP outperforms the recent methods (*e.g.*, GPM (replay-based), APD and WSN (expansion-based)) with a significant margin (*e.g.*, with

Method	CIFAR100-Split (with LeNet)				CIFAR100-SuperClass (with LeNet)			
	5%	25%	50%	100%	5%	25%	50%	100%
STL	45.13 ± 0.04	59.04 ± 0.03	64.38 ± 0.06	69.55 ± 0.06	43.76 ± 0.68	56.09 ± 0.07	60.06 ± 0.06	64.47 ± 0.05
MTL	44.95 ± 0.11	60.21 ± 0.28	65.65 ± 0.20	69.70 ± 0.28	40.43 ± 0.15	49.88 ± 0.27	53.83 ± 0.27	55.62 ± 0.41
L2	37.15 ± 0.21	48.86 ± 0.28	53.35 ± 0.34	58.09 ± 0.43	34.03 ± 0.08	43.40 ± 0.27	46.10 ± 0.28	48.75 ± 0.24
EWC	37.76 ± 0.20	50.09 ± 0.38	55.65 ± 0.40	60.53 ± 0.26	33.70 ± 0.32	44.02 ± 0.39	47.35 ± 0.47	49.97 ± 0.39
BN	37.60 ± 0.17	50.70 ± 0.28	54.79 ± 0.28	60.34 ± 0.40	36.76 ± 0.14	48.20 ± 0.16	51.43 ± 0.16	55.44 ± 0.19
BE	37.63 ± 0.15	51.13 ± 0.3	55.37 ± 0.28	61.09 ± 0.33	37.05 ± 0.20	48.48 ± 0.14	51.78 ± 0.17	55.97 ± 0.17
APD	36.60 ± 0.14	54.59 ± 0.07	59.71 ± 0.03	66.54 ± 0.03	32.81 ± 0.29	49.00 ± 0.06	52.64 ± 0.19	60.54 ± 0.23
APDfix	35.66 ± 0.33	54.62 ± 0.11	59.86 ± 0.24	66.64 ± 0.14	24.27 ± 0.22	48.71 ± 0.11	53.42 ± 0.12	61.47 ± 0.16
IBWPF	38.35 ± 0.26	47.87 ± 0.25	53.46 ± 0.13	57.13 ±0.15	33.09 ± 0.50	51.32 ± 0.27	52.52 ± 0.26	55.98 ± 0.33
GPM	32.86 ± 0.35	51.61 ± 0.22	57.60 ± 0.19	64.49 ± 0.10	34.88 ± 0.30	47.31 ± 0.51	51.23 ± 0.55	57.91 ± 0.28
WSN	37.01 ± 0.63	55.21 ± 0.59	61.56 ± 0.42	66.56 ± 0.49	36.89 ± 0.49	52.42 ± 0.62	58.23 ± 0.38	61.81 ± 0.54
BMKP	42.36 ± 0.90	56.81 ± 1.05	62.87 ± 0.60	66.95 ± 0.53	37.26 ± 0.87	53.62 ± 0.59	57.76 ± 0.66	61.97 ± 0.19
CLR	36.46 ± 0.29	51.44 ± 0.35	57.00 ± 0.43	61.83 ± 0.60	37.93 ± 0.25	49.82 ± 0.56	53.86 ± 0.63	57.07 ± 0.60
PRD	31.58 ± 0.29	56.07 ± 0.22	59.61 ± 0.33	62.74 ± 0.45	33.34 ± 0.48	52.99 ± 0.28	55.85 ± 0.45	57.80 ± 0.50
HALRP	**45.09 ± 0.05**	**58.94 ± 0.09**	**63.61 ± 0.08**	**67.92 ± 0.17**	**43.84 ± 0.04**	**54.93 ± 0.04**	**58.68 ± 0.11**	**62.56 ± 0.30**

Table 4.2: Accuracies↑ on CIFAR100-Split/-SuperClass with different percentages of training data.

an improvement about 1% ∼ 3%). Especially, we also evaluated the performances of all the methods under different amounts of training data (*i.e.*, 5% ∼ 100%) on these two benchmarks. It was interesting to see that our proposed method had advantages in dealing with extreme cases with limited data. Compared to other methods, the performance margins become more significant with fewer training data. For example, on CIFAR100-Split, our proposed HALRP outperforms APD and WSN with an improvement of 1.28% and 1.36% respectively, but these margins will dramatically rise to ∼ 8% if we reduce the training set to 5% of the total data. These results showed that our method can work better in these extreme cases.

Especially, we also conducted experiments on the Omniglot-Ratation dataset to demonstrate the scalability of our method. We follow [1] to learn the 100 tasks under the default sequential order. The average accuracy was shown in Table 4.3b. We demonstrate that our method is applicable to a large number of tasks and can still achieve comparable performance with limited model increment.

On Five-dataset, we adopted two types of neural network, *i.e.*, AlexNet and ResNet18, to verify the effectiveness of HALRP under different backbones. We can observe that our

Method	P-MNIST LeNet Acc.↑	MOPD↓	AOPD↓
STL	98.24±0.01	0.15	0.09
MTL	96.70±0.07	1.58	0.81
L2	79.14±0.70	29.66	18.94
EWC	81.69±0.86	21.51	12.16
BN	81.04±0.15	19.77	8.18
BE	83.80±0.08	16.76	6.88
APD	97.94±0.02	0.25	0.16
APDfix	97.99±0.01	0.10	0.11
GPM	96.69±0.02	0.45	0.27
WSN	97.91±0.02	0.36	0.22
BMKP	97.08±0.01	3.21	1.04
CLR	88.55±0.20	14.97	7.88
PRD	83.16±0.17	7.91	6.16
HALRP	**98.10±0.03**	0.47	0.24

(a) Results on P-MNIST.

Method	Omniglot-Rotation LeNet Acc.↑	MOPD↓	AOPD↓
STL	80.93±0.18	20.83	3.42
MTL	93.95±0.11	6.25	2.11
L2	69.86±1.23	17.23	6.96
EWC	69.75±1.28	21.39	7.02
BN	77.08±0.86	14.41	5.58
BE	78.24±0.69	17.17	5.53
APD(⋆)	81.60±0.53	8.19	3.78
APDfix	78.14±0.12	6.53	2.63
GPM	80.41±0.16	28.33	13.02
WSN	82.55±0.44	17.09	7.57
BMKP	81.12±2.71	26.53	16.17
CLR	72.75±1.41	24.30	12.27
PRD	74.49±2.78	49.17	18.44
HALRP	**83.08±0.73**	10.36	3.91

(b) Results on Omniglot.

Method	Five-dataset AlexNet Acc.↑	MOPD↓	AOPD↓	ResNet-18 Acc.↑	MOPD↓	AOPD↓
STL	89.32±0.06	0.74	0.30	94.24±0.05	0.67	0.24
MTL	88.02±0.18	2.08	0.68	93.82±0.06	0.67	0.30
L2	78.24±2.00	35.11	15.35	85.94±2.79	37.03	12.72
EWC	78.44±2.20	33.29	13.18	86.32±2.80	33.84	11.80
BN	82.35±2.45	34.83	12.56	88.36±2.21	30.22	9.55
BE	82.91±2.37	33.91	11.63	88.75±2.14	29.22	8.97
APD	83.70±0.90	4.80	3.45	92.18±0.28	3.50	1.54
APDfix	84.03±1.24	5.50	3.66	91.91±0.48	6.74	1.98
GPM	87.27±0.61	4.54	1.88	88.52±0.28	6.97	2.82
WSN	86.74±0.40	8.54	2.89	92.58±0.39	4.62	1.21
BMKP	84.03±0.55	9.32	3.07	92.57±0.65	9.08	2.13
CLR	86.68±1.41	19.78	7.08	90.04±1.04	14.05	4.51
PRD	74.74±0.69	17.53	9.23	88.45±0.93	14.17	5.37
HALRP	**88.81±0.31**	4.28	1.31	**93.39±0.30**	4.39	1.27

(c) Results on Five-dataset with different backbones.

Table 4.3: Performance on P-MNIST, Five-dataset, Omniglot-Rotation. We ran experiments with five different task orders generated by different seeds. As for Omniglot-Rotation, we follow [1] to show the scalability under the original sequential order. Acc.↑ refers to the empirical accuracy, MOPD↓ and AOPD↓ refer to the task order robustness as discussed in Section 4.3. Results with ⋆ are from [2].

	TinyImageNet 20-split						TinyImageNet 40-split					
	AlexNet			ResNet18			AlexNet			ResNet18		
Method	Acc.↑	MODP↓	AODP↓	Acc.↑	MODP↓	AODP↓	Acc.↑	MODP↓	AODP↓	Acc.↑	MODP↓	AODP↓
STL	66.78±0.18	6.20	3.43	67.00±0.30	5.87	2.87	74.65±0.17	8.16	4.36	74.08±0.25	8.13	4.08
MTL	71.23±0.54	6.87	3.42	73.64±0.46	6.94	3.31	78.80±0.25	7.74	3.92	80.05±0.22	9.07	4.04
L2	56.33±0.22	11.93	5.72	60.80±0.56	8.27	4.23	63.47±1.35	16.27	7.42	65.59±1.03	14.80	7.00
EWC	56.55±0.22	10.93	5.53	60.88±0.56	7.54	4.63	64.11±1.36	17.87	6.91	66.54±0.94	14.67	6.99
BN	57.20±0.11	12.07	5.37	61.03±0.65	9.73	4.24	64.04±1.15	19.07	7.23	66.17±1.75	15.33	6.92
BE	57.62±0.41	11.80	4.98	61.52±0.68	7.00	4.09	64.70±1.08	23.47	6.97	66.77±1.61	16.94	7.07
APD	**67.26±0.38**	12.00	5.43	68.76±0.58	9.86	4.33	73.88±0.46	8.53	5.04	73.85±0.40	11.46	5.57
APDfix	63.59±0.25	5.40	3.68	67.69±0.41	6.54	3.64	67.91±1.33	23.47	8.95	58.11±2.11	26.93	11.70
GPM	60.84±0.32	11.00	4.44	48.09±1.07	9.27	4.80	70.14±0.38	10.90	4.52	43.40±7.68	48.80	38.80
WSN	65.73±0.21	5.06	3.31	68.27±0.47	8.40	4.52	73.46±1.27	8.00	4.23	75.29±0.46	12.27	4.75
BMKP	65.01±0.41	5.87	3.37	67.45±0.59	10.60	6.69	73.57±0.21	9.60	4.31	74.84±0.63	12.90	4.95
CLR	57.29±0.39	8.80	4.42	61.77±0.47	10.27	5.42	65.22±0.49	9.86	5.74	67.59±0.92	16.54	9.30
PRD	46.49±0.30	15.40	8.33	49.65±0.57	19.33	11.59	53.54±0.45	19.74	10.02	63.78±0.59	20.27	7.93
HALRP	66.68±0.20	5.60	3.43	**70.09±0.29**	4.67	3.20	**74.01±0.25**	7.47	4.12	**75.53±0.50**	7.87	4.48

Table 4.4: Results on TinyImageNet with different backbones. We ran experiments with five different task orders generated by different seeds. Acc.↑ refers to the empirical accuracy, MOPD↓ and AOPD↓ refer to the task order robustness as discussed in Section 4.3.

method consistently outperforms the other methods under different backbones, indicating the applicability and flexibility for different model choices. Compared with the analyses in the following part, we can conclude that our method is also robust under different orders of tasks.

On TinyImageNet dataset, we evaluated our methods on two settings, one with 20-split and another with 40-split, and the results are posted in Table 4.4. According to the empirical results, we can conclude that our proposed method can perform well on this challenging dataset, with a better trade-off between the average accuracy and task order robustness. Compared to the previous methods APD and APDfix that aim to address the issue of task order robustness, our HALRP can perform well in a more consistent manner. Furthermore, our methods also have advantages regarding computational efficiency and memory consumption that we will discuss later in the following sections.

Additionally, we also provide comparisons with IRU [3] that is also based on low-rank decomposition. We note that the official code of IRU only contains a demo for multi-layer perceptron (MLP), and no complete code for the convolutional network (even LeNet) can be found. Due to this limitation, we cannot implement IRU under our setting. To make fair

	PMNIST-20 (MLP)	CIFAR100-20 (ResNet18)	CIFAR100-20 (MLP)
Multitask(⋆)	96.8	70.2	16.4
IRU(⋆)	85.60 ± 0.15	68.46 ± 2.52	65.90 ± 2.16
HALRP	92.02 ± 0.04	73.71 ± 0.87	67.21 ± 0.68

Table 4.5: Accuracies under the IRU [3] setting. Results with ⋆ are reported in [3].

comparisons, we re-implement our method under the setting of IRU [3]. We describe the setting as follows:

(1) **Dataset Protocol**: [3] generated 20 random tasks on PMNIST, rather than 10 tasks in this work. Moreover, [3] randomly divided the CIFAR100 dataset into 20 tasks, rather than the superclass-based splitting in this thesis. Thus, we denote the two datasets in [3] as "PMNIST-20" and "CIFAR100-20", to distinguish them from the "PMNIST" and "CIFAR100-SuperClass" used in the common setting.

(2) **Model Architecture**: Apart from the architectures we introduced before, we also follow the MLP used in [3], a three-layer (fully-connected) multilayer perceptron with 256 hidden nodes.

The results of our method and IRU are listed in Table 4.5. We can conclude that our method achieved a large performance gain compared to IRU.

In the following parts, we will further investigate the empirical performance under different aspects, i.e., task order robustness, forgetting statistics, model growth, and time complexity. We will show that our proposed method can achieve a better trade-off among these realistic metrics apart from the average accuracy.

4.3 Robustness on Task Orders

We evaluate the robustness of these algorithms under different task orders. Following the protocol of [2], we assessed the task order robustness with the **Order-normalized Performance Disparity** (OPD) metric, which is computed as the disparity between the perfor-

	CIFAR100-Split (with LeNet)								CIFAR100-SuperClass (with LeNet)							
	5%		25%		50%		100%		5%		25%		50%		100%	
Method	MOPD↓	AOPD↓	MOPD↓	AOPD↓	MOPD↓	AOPD↓	MOPD↓	AOPD↓	MOPD↓	AOPD↓	MOPD↓	AOPD↓	MOPD↓	AOPD↓	MOPD↓	AOPD↓
STL	1.96	1.38	3.44	2.41	3.68	2.57	4.38	3.13	2.52	1.48	3.64	1.44	2.52	1.48	2.64	1.45
MTL	1.44	0.93	1.34	0.87	1.66	0.71	1.54	0.84	6.96	2.66	12.04	4.89	13.96	5.74	13.48	6.29
L2	11.66	6.13	13.14	6.96	15.02	7.50	15.18	7.15	9.92	4.55	13.24	6.43	13.32	7.04	14.44	6.82
EWC	11.92	6.07	13.20	6.85	13.78	6.96	13.44	6.48	13.24	5.60	10.92	6.53	13.48	8.03	21.60	8.32
BN	11.70	5.95	13.28	7.37	12.40	6.91	13.92	7.90	8.52	3.35	11.20	3.99	11.64	4.05	11.40	4.25
BE	12.12	5.47	12.92	6.13	9.28	5.56	8.50	5.35	9.56	3.46	11.88	4.00	11.80	3.88	10.84	4.03
APD	11.42	7.21	6.48	3.77	8.00	4.10	8.88	4.07	26.56	12.98	8.64	4.39	8.28	4.48	6.72	3.26
APDfix	6.64	4.39	8.32	4.94	8.92	5.54	7.40	4.21	9.04	4.90	10.28	5.68	8.56	5.32	6.04	2.70
IBPWF	4.30	2.84	4.60	2.73	5.40	3.07	3.68	2.68	14.84	7.45	4.44	2.45	5.36	2.86	5.52	3.38
GPM	11.14	6.37	6.32	4.22	6.32	3.69	2.28	1.348	9.32	4.46	10.00	5.53	8.84	5.89	7.68	4.47
WSN	4.16	2.57	4.42	2.71	4.2	2.62	3.56	2.39	5.08	3.14	4.76	3.192	4.00	2.16	3.76	2.36
BMKP	12.98	7.63	13.22	6.50	6.52	4.30	8.34	3.35	11.72	6.03	13.32	5.27	11.08	4.43	3.72	1.99
CLR	13.50	7.14	12.77	6.13	8.90	5.82	9.37	5.34	10.00	4.01	7.67	3.84	6.60	3.71	9.00	4.06
PRD	5.80	3.80	4.60	2.66	4.16	2.63	5.20	3.16	7.47	4.28	8.33	3.95	5.33	3.29	5.00	2.86
HALRP	2.56	1.34	2.58	1.44	2.34	1.71	3.90	2.56	2.96	1.65	3.40	1.91	4.48	1.65	4.34	1.96

Table 4.6: Task order robustness evaluation on CIFAR100-Split/SuperClass with different amounts of training data.

mance A_t of task t under R different task orders:

$$\text{OPD}_t \triangleq \max\{A_t^1, \ldots, A_t^R\} - \min\{A_t^1, \ldots, A_t^R\} \tag{4.3}$$

In our experiments, we take five different task orders (i.e., $R = 5$. See Appendix E.1).

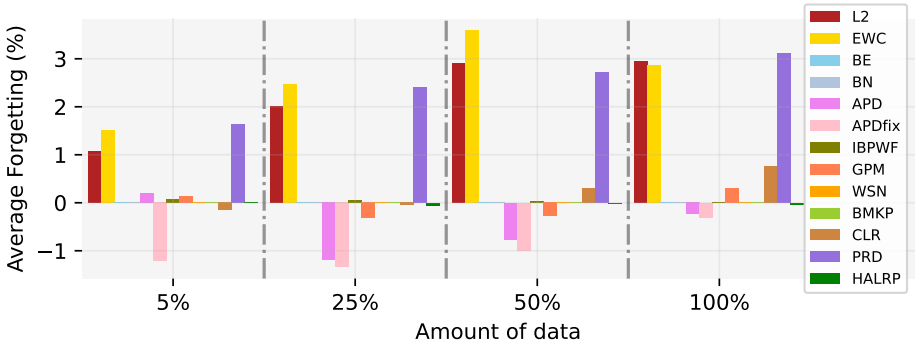
The maximum OPD (MOPD) and average OPD (AOPD) among all tasks are defined by

$$\begin{aligned} \text{MOPD} &\triangleq \max\{\text{OPD}_0, \ldots, \text{OPD}_{T-1}\} \\ \text{AOPD} &\triangleq \frac{1}{T}\sum_{t=0}^{T-1} \text{OPD}_t \end{aligned} \tag{4.4}$$

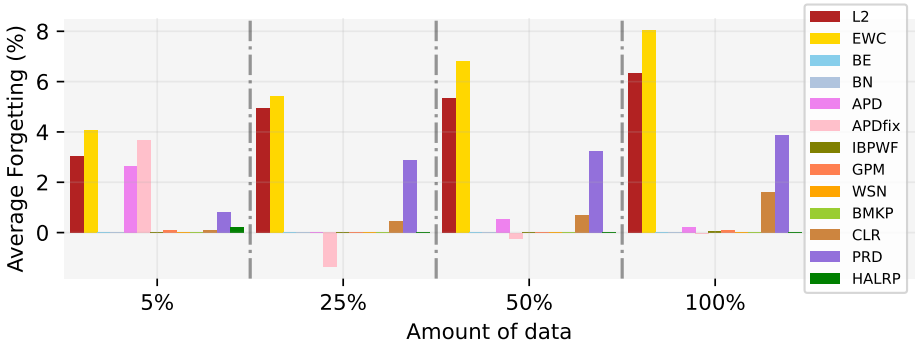
respectively.

Thus, smaller AOPD and MOPD indicate better task-order robustness. For CIFAR100-Split and SuperClass datasets, we adopted different orders by following [2]. For the P-MNIST and Five-dataset, we report the averaged results over five different task orders. The results on MOPD and AOPD on P-MNIST, Omniglot-Rotation, Five-dataset, TinyImageNet, and CIFAR100-Splits/-SuperClass are reported in Table 4.3a, Table 4.3b, Table 4.3c, Table 4.4 and Table 4.6, respectively. According to these results, we can conclude that our method can achieve a better accuracy-robustness trade-off, compared to the baseline APD.

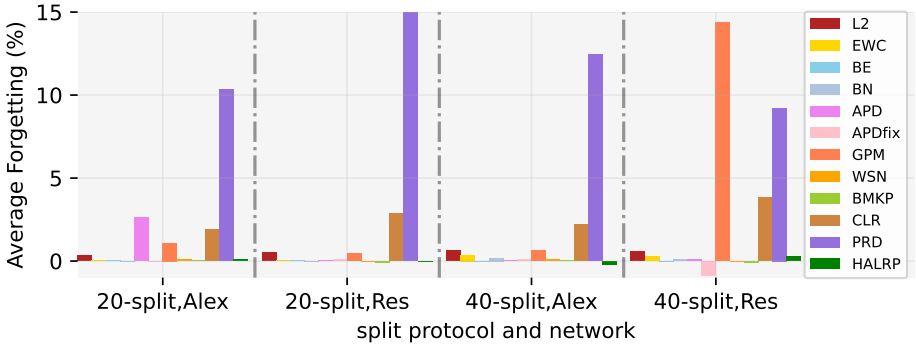
The reason why our proposed HALRP achieved better task-order robustness can be credited by the residual learning with respect to the model parameters. Specifically, the parameters for each task $\mathcal{T}_t$ was represented as $\mathbf{W}_t = \mathbf{R}_t\mathbf{W}^{\text{base}}\mathbf{S}_t + \mathbf{U}_t^{(k)}\Sigma_t^{\text{base}}(\mathbf{V}_t^{(k)})^\top$, including the parameters inherited from the task $\mathcal{T}_0$ and the residual private parameters of the task $\mathcal{T}_t$. This form was leveraged to benefit the performance robustness when the tasks come with different orders.



(a) Forgetting on 10 tasks in CIFAR100-Splits



(b) Forgetting on 20 tasks in CIFAR100-SuperClass



(c) Forgetting on TinyImageNet. "Alex" for AlexNet and "Res" for ResNet18

Figure 4.1: Average Forgetting Statistics on CIFAR100-Splits/-SuperClass/TinyImageNet Datasets

4.4 Handling the Catastrophic Forgetting

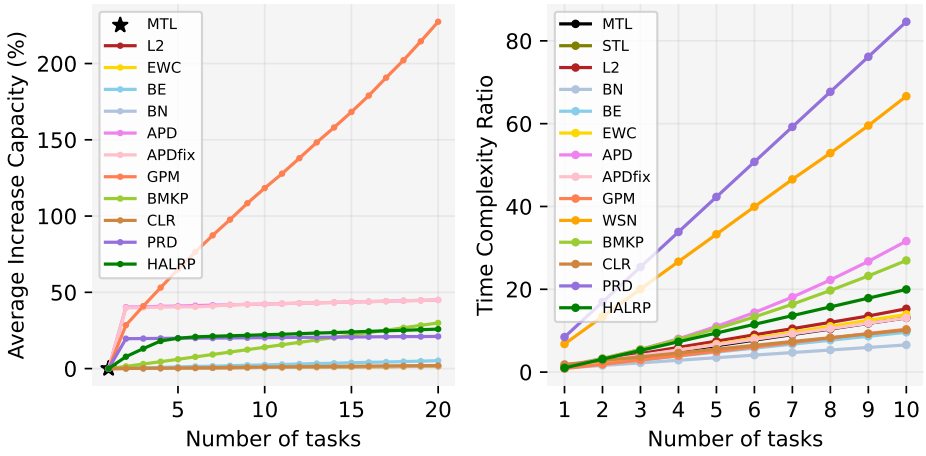
We then evaluated the abilities of different methods for overcoming catastrophic forgetting. We illustrate the average forgetting (BWT) on the CIFAR100 Split/SuperClass dataset under different amounts of data in Fig. 4.1. Our proposed method can effectively address the forgetting issue, comparable with some state-of-the-art methods like WSN. We further provide detailed forgetting statistics within five different task orders in Appendix D.2.

4.5 Model Increment Analysis

Another potential issue in continual learning is the model size growth as the number of tasks increases. To compare increased model capacities among different methods, we visualize *the ratio of increased parameters* w.r.t to base model on the sequential 20 tasks of CIFAR100-SuperClass dataset in Fig. 4.2a. We can observe that our proposed HALRP can better control the model capacity increment. After the first four tasks, the model parameters increased by around 19% and grow slowly during the last sixteen tasks (finally $\leq$ 28%). In contrast, the model parameters of APD grew quickly by around 40% in the first two or three tasks and kept a high ratio during the following tasks.

4.6 Computational Efficiency

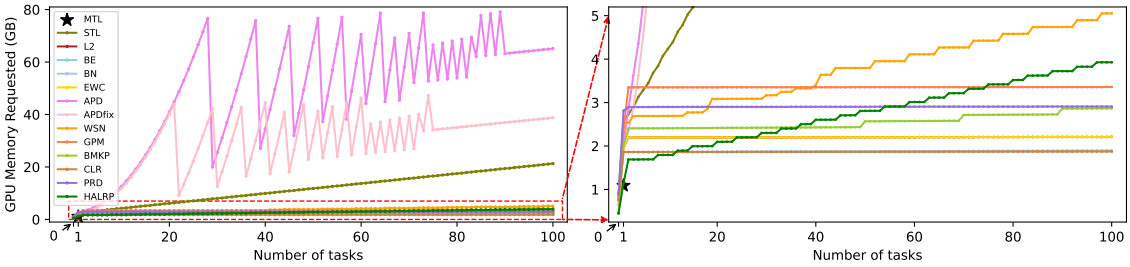
We visualize the average time complexity ratio on the ten tasks of the PMNIST dataset in Fig. 4.2b. The time complexity ratio is computed by dividing the accumulated time across the tasks w.r.t. the time cost on the first task of single-task learning. We observe that our proposed method is also computationally efficient with limited time consumption compared to most baselines, with a better trade-off between performance and efficiency. Especially, our method needs less training time compared to WSN, BMKP and APD.



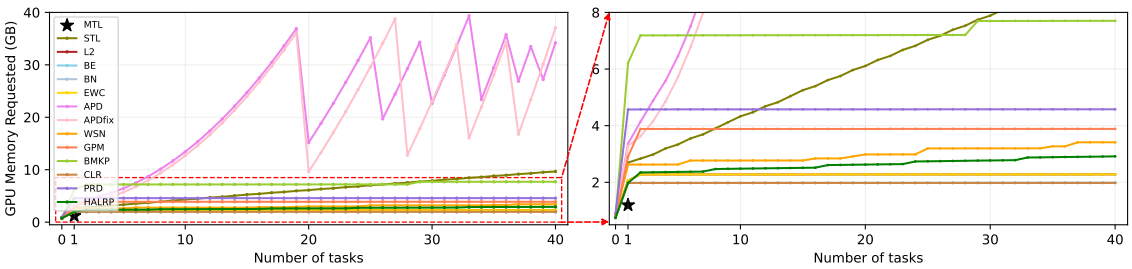
(a) (b)

Figure 4.2: (a) Average Capacity Increment ratio on CIFAR100-SuperClass w.r.t. the base model. (b) Average Time Complexity Ratio on PMNIST.

The major reason that should account for the inefficiency of WSN is that this method needs to optimize a weight score for each model parameter and then generate binary masks by locating a certain sparsity quantile among the weight scores in each layer, which is very time-consuming. Especially, we can observe a non-linear increase in behavior for BMKP and APD in Fig. 4.2b. In BMKP, the pattern basis of the core knowledge space increases along the sequential tasks and then the knowledge projection between two memory levels will cost more time as the tasks pass. A severe drawback of APD method is that when the task t arrives for learning, it needs to recover the parameters $\theta_i^\star$ for all previous tasks $i = 0, 1, ..., t-1$ and then apply the regularization $\Sigma_{i=0}^{t-1}||\theta_t - \theta_i^\star||_2^2$ between the weights of current task (i.e., θ_t) and each previous task (i.e., $\theta_i^\star$). This defect becomes more time-consuming as more tasks arrive. Besides, an extra k-means clustering process is needed for the hierarchical knowledge consolidation among tasks in APD. The difference of computational efficiency becomes more significant on the challenging scenarios with more tasks and complex network architectures. As for PRD, it is inefficient as this method converges slower than other methods, due to the reason that it adopted supervised contrastive loss instead of cross-entropy loss.



(a) GPU memory usage for Omniglot-Rotation with LeNet



(b) GPU memory usage for TinyImageNet 40-split with AlexNet

Figure 4.3: Empirical statistics of GPU memory usage. The local zone with a red rectangle on the left is zoomed in on the right.

In addition to time efficiency, we also provided some quantitative comparisons to support the memory efficiency of our proposed HALRP. To compare the GPU memory overhead among different methods, we visualize the amount of GPU memory requested by each method along the increase of task numbers during the training process. Specifically, we tracked the GPU memory usage of two groups of representative and challenging experimental scenarios:

(1) Omniglot Rotation dataset with LeNet, which has 100 tasks in total;

(2) TinyImageNet dataset with AlexNet, which has the largest number of parameters (i.e., #parameters $\approx$ 62 million, FLOPS $\approx$ 724 million in AlexNet) and the second largest number of tasks (i.e., 40 tasks).

To make fair comparisons, we adopted the same hyperparameters (i.e., batch size, number of threads used in the data loader) for all the methods under each scenario, and moved all training-related operations (i.e., Singular Value Decomposition) onto the GPU devices. To

monitor the GPU memory usage, we embedded some snippets with the Python package `GPUtil`[2] into the training code. The amount of GPU memory requested during the training process is shown in the following Figure 4.3.

According to the visualization results in Figure 4.3, we can observe that our proposed HALRP is still memory efficient compared to other baseline methods under these two challenging scenarios. Specifically, under the scenario for Omniglot Rotation datasets with LeNet (Fig. 4.3a), we can observe that the amount of GPU memory requested by HALRP increased along the tasks but finally did not exceed 4GB for these 100 tasks. In contrast, APD and APDfix needed more GPU memory during training (i.e., maximum 80GB for APD and 45GB for APDfix), making the related experiments hard to be reproduced unless on some specific GPUs like NVIDIA A100 80G. As for TinyImageNet dataset with AlexNet, our proposed HALRP only requested about 3GB GPU memory at the end of the 40-th task, which is much lower than APD and APDfix that need at least 40GB memory during training, as well as BMKP that needs up to 8GB.

To summarize, our proposed method HALRP will not introduce heavy memory overhead. In contrast, it is memory-efficient. The empirical results show that the requirement of training is easy to be satisfied and the scalability under the above challenging scenarios is easy to achieve.

4.7 Ablation Studies

In this part, we provide ablation studies about some hyperparameters in our method.

Ablation study about the rank selection We report the additional studies to evaluate the impacts of hyperparameter selection on the rank selection process. We conduct the following ablations: (1) varying the warm-up epochs n_w, (2) varying the loss approximation rate

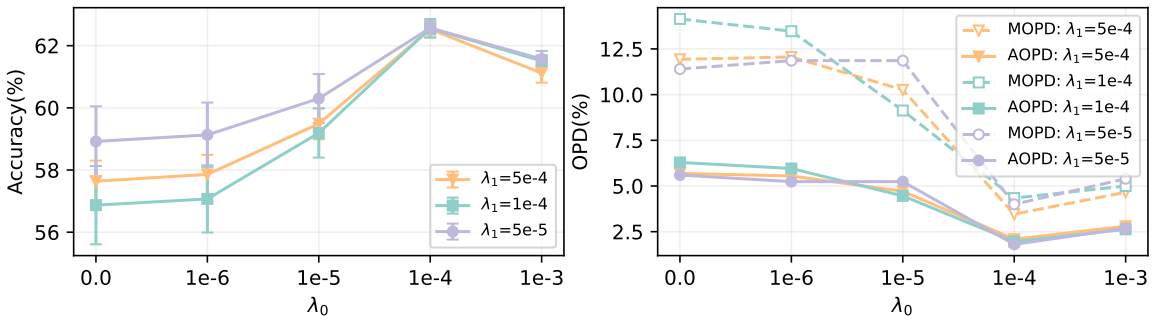
[2] https://pypi.org/project/GPUtil/

Ablation	Acc.↑	Size↓
$\alpha = 0.90, n_w = 1$	68.39 ± 0.13	0.234
$\alpha = 0.90, n_w = 2$	67.93 ± 0.19	0.234
$\alpha = 0.90, n_w = 3$	67.38 ± 0.12	0.234
$\alpha = 0.90, n_w = 4$	67.04 ± 0.08	0.234
$\alpha = 0.95, n_w = 1$	68.09 ± 0.15	0.234
$\alpha = 0.75, n_w = 1$	68.26 ± 0.11	0.217
$\alpha = 0.60, n_w = 1$	66.87 ± 0.23	0.165
$\alpha = 0.45, n_w = 1$	66.08 ± 0.14	0.100
LRP.	67.49± 0.13	0.234
Random LRP.	61.76 ± 0.18	0.235

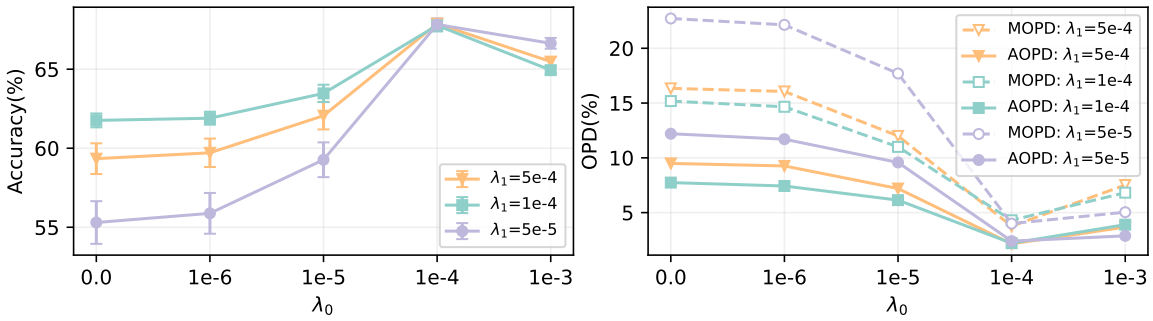
Table 4.7: Ablation studies. "Acc.↑" refers to the accuracy; "Size↓" refers to the relative increment size ratio.

α, (3) **LRP**: omitting the importance estimation (step 6 of Algorithm 1), and (4) **Random LRP**: replacing step 6 of Algorithm 1 with a random decomposition. The relevant results on CIFAR100-Splits are depicted in Table 4.7. We can observe that the performance will not significantly rise as n_w increases, which indicates that only one epoch is enough for warm-up training. When decreasing the approximate rate α, the accuracy will decay. Furthermore, if we apply the random decomposition of the model, the accuracy will sharply drop, indicating the necessity of our Hessian aware-decomposition procedure.

Effect of the regularization coefficients λ_0 and λ_1 To investigate the effects of the regularization coefficients introduced in Eq. 3.17, we further conducted experiments on two datasets: CIFAR100-SuperClass and CIFAR100-Splits. We adopted values from $\{0, 1e-6, 1e-5, 1e-4, 1e-3\}$ for λ_0 and $\{5e-4, 1e-4, 5e-5\}$ for λ_1. For each combination, we reported MOPD/AOPD for task order robustness as well as the average accuracy on the above two datasets, and the results are illustrated in Fig. 4.4. We can see that the regularization coefficients λ_0 and λ_1 have potential effects on the final accuracy and task order robustness. In our experiments, we choose the optimal hyperparameters by validation. Furthermore, some selected hyperparameters are also applied to other baselines to realize fair comparisons, e.g., we set L_2-regularizer coefficient $\lambda_1 = 1e-4$ for all other methods on CIFAR100-SuperClass and CIFAR100-Splits. See Appendix E.3 for more details.



(a) Accuracy↑ (left) and MOPD↓/AOPD↓ (right) on CIFAR100-SuperClass (20 tasks)



(b) Accuracy↑ (left) and MOPD↓/AOPD↓ (right) on CIFAR100-Splits (10 tasks)

Figure 4.4: Effect of regularization coefficients λ_0 and λ_1.

Chapter 5

5 Conclusion and Future Work

5.1 Conclusion

In this work, we propose a low-rank perturbation method for continual learning. Specifically, we approximate the task-adaptive parameters with low-rank decomposition by formulating the model transition along the sequential tasks with parameter transformations. We theoretically show the quantitative relationship between the Hessian and the proposed low-rank approximation, which leads to a novel Hessian-aware framework that enables the model to automatically select ranks by the relevant importance of the perturbation to the model's performance. The extensive experimental results show that our proposed method performs better on the robustness of different task orders and the ability to address catastrophic forgetting issues.

5.2 Future Work

There are still some under-explored topics about applying our method to complex continual learning scenarios. For example, this thesis mainly considers the vision datasets commonly-used in continual learning, neglecting the emerging text data and multi-modality data in the community. How to improve the training and inference efficiency when adopting our method on Transformer-based large language models is worthy of exploring. Besides, some novel learning paradigms relevant to large language models, e.g., prompt tuning, can also be considered when plugging our current work into the relevant models. We will explore these problems in our future work.

References

[1] H. Kang, R. J. L. Mina, S. R. H. Madjid, J. Yoon, M. Hasegawa-Johnson, S. J. Hwang, and C. D. Yoo, "Forget-free continual learning with winning subnetworks," in *International Conference on Machine Learning*. PMLR, 2022, pp. 10 734–10 750.

[2] J. Yoon, S. Kim, E. Yang, and S. J. Hwang, "Scalable and order-robust continual learning with additive parameter decomposition," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[3] R. Hyder, K. Shao, B. Hou, P. Markopoulos, A. Prater-Bennette, and M. S. Asif, "Incremental task learning with incremental rank updates," in *European Conference on Computer Vision*. Springer, 2022, pp. 566–582.

[4] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3366–3385, 2021.

[5] G. Saha, I. Garg, and K. Roy, "Gradient projection memory for continual learning," in *International Conference on Learning Representations*, 2021.

[6] M. Farajtabar, N. Azizan, A. Mott, and A. Li, "Orthogonal gradient descent for continual learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 3762–3773.

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[8] S. Jung, H. Ahn, S. Cha, and T. Moon, "Continual learning with node-importance based adaptive group sparse regularization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3647–3658, 2020.

[9] M. K. Titsias, J. Schwarz, A. G. d. G. Matthews, R. Pascanu, and Y. W. Teh, "Functional regularisation for continual learning with gaussian processes," *arXiv preprint arXiv:1901.11356*, 2019.

[10] S. Wang, X. Li, J. Sun, and Z. Xu, "Training networks in null space of feature covariance for continual learning," in *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, 2021, pp. 184–193.

[11] Y. Kong, L. Liu, H. Chen, J. Kacprzyk, and D. Tao, "Overcoming catastrophic forgetting in continual learning by exploring eigenvalues of hessian matrix," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2023.

[12] N. Mehta, K. Liang, V. K. Verma, and L. Carin, "Continual learning using a bayesian nonparametric dictionary of weight factors," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 100–108.

[13] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[14] Y. Wen, D. Tran, and J. Ba, "Batchensemble: an alternative approach to efficient ensemble and lifelong learning," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[15] A. Chaudhry, N. Khan, P. Dokania, and P. Torr, "Continual learning in low-rank orthogonal subspaces," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9900–9911, 2020.

[16] Y. Yang, Z.-Q. Sun, H. Zhu, Y. Fu, Y. Zhou, H. Xiong, and J. Yang, "Learning adaptive embedding considering incremental class," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 3, pp. 2736–2749, 2023.

[17] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[18] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *Proceedings of International Conference on Learning Representations*, 2017.

[19] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3925–3934.

[20] A.-A. Liu, H. Lu, H. Zhou, T. Li, and M. Kankanhalli, "Balanced class-incremental 3d object classification and retrieval," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–13, 2023.

[21] Y. Ge, Y. Li, S. Ni, J. Zhao, M.-H. Yang, and L. Itti, "Clr: Channel-wise lightweight reprogramming for continual learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 18 798–18 808.

[22] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro, "Learning to learn without forgetting by maximizing transfer and minimizing interference," *arXiv preprint arXiv:1810.11910*, 2018.

[23] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[24] B. Zhang, Y. Guo, Y. Li, Y. He, H. Wang, and Q. Dai, "Memory recall: A simple neural network training framework against catastrophic forgetting," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2010–2022, 2022.

[25] H. Chen, Y. Wang, and Q. Hu, "Multi-granularity regularized re-balancing for class incremental learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 7263–7277, 2023.

[26] G. Sun, Y. Cong, Y. Zhang, G. Zhao, and Y. Fu, "Continual multiview task learning via deep matrix factorization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 139–150, 2021.

[27] S. Ho, M. Liu, L. Du, L. Gao, and Y. Xiang, "Prototype-guided memory replay for continual learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[28] L. Wang, B. Lei, Q. Li, H. Su, J. Zhu, and Y. Zhong, "Triple-memory networks: A brain-inspired method for continual learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 1925–1934, 2021.

[29] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," *Advances in neural information processing systems*, vol. 30, 2017.

[30] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-gem," *International Conference on Learning Representations*, 2019.

[31] C. Tai, T. Xiao, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[32] Y. Idelbayev and M. A. Carreira-Perpinán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8049–8059.

[33] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavskỳ, V. Glukhov, I. Oseledets, and A. Cichocki, "Stable low-rank tensor decomposition for compression of convolutional neural network," in *European Conference on Computer Vision*. Springer, 2020, pp. 522–539.

[34] L. Wang, M. Rege, M. Dong, and Y. Ding, "Low-rank kernel matrix factorization for large-scale evolutionary clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 1036–1050, 2012.

[35] X. Zhu, S. Zhang, Y. Li, J. Zhang, L. Yang, and Y. Fang, "Low-rank sparse subspace for spectral clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 8, pp. 1532–1543, 2019.

[36] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017.

[38] L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li, "Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning," *arXiv preprint arXiv:2308.03303*, 2023.

[39] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, "Adaptive budget allocation for parameter-efficient fine-tuning," in *The Eleventh*

International Conference on Learning Representations, 2023. [Online]. Available: https://openreview.net/forum?id=lq62uWRJjiY

[40] B. Zi, X. Qi, L. Wang, J. Wang, K.-F. Wong, and L. Zhang, "Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices," *arXiv preprint arXiv:2309.02411*, 2023.

[41] J. Li, Y. Lai, R. Wang, C. Shui, S. Sahoo, C. X. Ling, S. Yang, B. Wang, C. Gagné, and F. Zhou, "Hessian aware low-rank perturbation for order-robust continual learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 11, pp. 6385–6396, 2024.

[42] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[43] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Proceedings of 31st Conference on Neural Information Processing Systems*, 2017.

[44] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

[45] Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq-v2: Hessian aware trace-weighted quantization of neural networks," *Advances in neural information processing systems*, vol. 33, pp. 18 518–18 529, 2020.

[46] F. Kunstner, P. Hennig, and L. Balles, "Limitations of the empirical fisher approximation for natural gradient descent," *Advances in neural information processing systems*, vol. 32, 2019.

[47] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[48] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach, "Uncertainty-guided continual learning with bayesian neural networks," *arXiv preprint arXiv:1906.02425*, 2019.

[49] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[50] Y. Bulatov, "Notmnist dataset. google (books/ocr)," Tech. Rep.[Online]. Available: http://yaroslavvb. blogspot. it/2011/09 . . . , Tech. Rep., 2011.

[51] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[52] N. Yuval, W. Tao, C. Adam, B. Alessandro, W. Bo, and N. Andrew Y., "Reading digits in natural images with unsupervised feature learning," in *In Proceedings of NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. PMLR, 2011, pp. 522–539.

[53] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[55] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, 2022.

[56] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[57] W. Sun, Q. Li, J. Zhang, W. Wang, and Y.-a. Geng, "Decoupling learning and remembering: A bilevel memory framework with knowledge projection for task-incremental learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 20 186–20 195.

[58] N. Asadi, M. Davari, S. Mudur, R. Aljundi, and E. Belilovsky, "Prototype-sample relation distillation: towards replay-free continual learning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 1093–1106.

[59] V. K. Verma, K. J. Liang, N. Mehta, P. Rai, and L. Carin, "Efficient feature transformations for discriminative and generative continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 865–13 875.

Appendices

A Low-Rank Factorization for Matrix

Proof of Eq. 2.5. Here, we provide more background knowledge about the low-rank matrix factorization.

In this work, we leverage the Eckart–Young–Mirsky theorem [13] with the Frobenius norm. In this part, we provide background knowledge for the self-cohesion of the thesis.

Denote by $\mathbf{B} \in \mathbb{R}^{m \times n}$ a real (possibly rectangular) matrix. Suppose that $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the singular value decomposition (SVD) of $\mathbf{B}$, then, we can claim that the best rank k approximation ($k \leq \min\{m, n\}$) to $\mathbf{B}$ under the Frobenius norm $\|\cdot\|_F$ is given by $\mathbf{B}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$, where $\mathbf{u}_i$ and $\mathbf{v}_i$ denote the i^{th} column of $\mathbf{U}$ and $\mathbf{V}$, respectively. Then,

$$\|\mathbf{B} - \mathbf{B}_k\|_F^2 = \left\| \sum_{i=k+1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \right\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2$$

Thus, we need to show that if $\mathbf{A}_k = XY^\top$ where X and Y has k columns

$$\|\mathbf{B} - \mathbf{B}_k\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2 \leq \|\mathbf{B} - \mathbf{A}_k\|_F^2$$

.

By the triangle inequality, if $\mathbf{B} = \mathbf{B}' + \mathbf{B}''$ then $\sigma_1(\mathbf{B}) \leq \sigma_1(\mathbf{B}') + \sigma_1(\mathbf{B}'')$. Denote by $\mathbf{B}'_k$ and $\mathbf{B}''_k$ the rank k approximation to $\mathbf{B}'$ and $\mathbf{B}''$ by the SVD method, respectively. Then, for any $i, j \geq 1$,

$$\begin{aligned}\sigma_i(\mathbf{B}') + \sigma_j(\mathbf{B}'') &= \sigma_1(\mathbf{B}' - \mathbf{B}'_{i-1}) + \sigma_1(\mathbf{B}'' - \mathbf{B}''_{j-1}) \\ &\geq \sigma_1(\mathbf{B} - \mathbf{B}'_{i-1} - \mathbf{B}''_{j-1}) \\ &\geq \sigma_1(\mathbf{B} - \mathbf{B}_{i+j-2}) \\ &= \sigma_{i+j-1}(\mathbf{B})\end{aligned}$$

where the last inequality comes from the fact that $\text{rank}(\mathbf{B}'_{i-1} + \mathbf{B}''_{j-1}) \leq \text{rank}\,(\mathbf{B}_{i+j-2})$.

Since $\sigma_{k+1}(\mathbf{B}_k) = 0$, when $\mathbf{B}' = \mathbf{B} - \mathbf{A}_k$ and $\mathbf{B}'' = \mathbf{A}_k$ we conclude that for $i \geq 1$, $j = k + 1$

$$\sigma_i(\mathbf{B} - \mathbf{A}_k) \geq \sigma_{k+i}(\mathbf{B})$$

Therefore,

$$\|\mathbf{B} - \mathbf{A}_k\|_F^2 = \sum_{i=1}^{n} \sigma_i(\mathbf{B} - \mathbf{A}_k)^2 \geq \sum_{i=k+1}^{n} \sigma_i(\mathbf{B})^2 = \|\mathbf{B} - \mathbf{B}_k\|_F^2$$

Thus, we can get Eq. 2.5. □

B Proof to Theorem 1 and Discussion

We recall Theorem 1:

Theorem 1. *Assume that a neural network of L layers with vectorized weights* $(\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star)$ *that have converged to local optima, such that the first and second order optimally conditions are satisfied,* i.e.*, the gradient is zero, and the Hessian is positive semi-definite. Suppose a perturbation* $\Delta\boldsymbol{\omega}_1^\star$ *applied to the first layer weights, then we have the loss change*

$$\left|\mathcal{L}(\boldsymbol{\omega}_1^\star - \Delta\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star) - \mathcal{L}(\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star)\right| \leq \frac{1}{2}\|\mathbf{H}_1\|_F \cdot \|\Delta\boldsymbol{\omega}_1^\star\|_F^2 + o\left(\|\Delta\boldsymbol{\omega}_1^\star\|_F^2\right), \tag{3.12}$$

where $\mathbf{H}_1 = \nabla^2 \mathcal{L}(\boldsymbol{\omega}_1^\star)$ *is the Hessian matrix at only the variables of the first layer weights.*

Proof. Denote the gradient and Hessian of the first layer $\boldsymbol{\omega}_1^\star$ as $\mathbf{g}_1$ and $\mathbf{H}_1$. Through Taylor's expansion, we have

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\omega}_1^\star - \Delta\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star) - \mathcal{L}(\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star) \\
&\quad = -\mathbf{g}_1^T \Delta\boldsymbol{\omega}_1^\star + \frac{1}{2} \Delta\boldsymbol{\omega}_1^{\star T} \mathbf{H}_1 \Delta\boldsymbol{\omega}_1^\star + o(\|\Delta\boldsymbol{\omega}_1^\star\|_F^2).
\end{aligned}$$

Using the fact that the gradient is zero at the local optimum $\boldsymbol{\omega}_1^\star$ as well as the sub-additive and sub-multiplicative properties of Frobenius norm, we have,

$$\begin{aligned}
&\left|\mathcal{L}(\boldsymbol{\omega}_1^\star - \Delta\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star) - \mathcal{L}(\boldsymbol{\omega}_1^\star, \ldots, \boldsymbol{\omega}_L^\star)\right| \\
&\quad = |\frac{1}{2} \Delta\boldsymbol{\omega}_1^{\star T} \mathbf{H}_1 \Delta\boldsymbol{\omega}_1^\star + o(\|\Delta\boldsymbol{\omega}_1^\star\|_F^2)| \\
&\quad \leq \|\frac{1}{2} \Delta\boldsymbol{\omega}_1^{\star T} \mathbf{H}_1 \Delta\boldsymbol{\omega}_1^\star\|_F + o(\|\Delta\boldsymbol{\omega}_1^\star\|_F^2) \\
&\quad \leq \frac{1}{2} \|\mathbf{H}_1\|_F \|\Delta\boldsymbol{\omega}_1^\star\|_F^2 + o(\|\Delta\boldsymbol{\omega}_1^\star\|_F^2).
\end{aligned}$$

□

Furthermore, Algorithm 1 implies that the Hessian information can be used to quantitatively measure the influences of low-rank perturbation on the model's empirical losses. In practice, we can approximate the Hessian by the negative empirical Fisher information [46]. This enables a dynamic scheme for the trade-off between the approximation error and computational efficiency. For a given loss approximation rate, the model can automatically select the rank for all the layers.

C Discussion on the Fine-tuning Objective on the New Task

In the proposed algorithm, we finally fine-tune the model on the new task with Eq. 12 in the manuscript, which is also listed below,

$$\min_{\mathbf{W}_t} \mathcal{L}(\mathbf{W}_t; \mathcal{D}_t) + \mathcal{L}_{\text{reg}}(\mathbf{W}_t) \tag{1}$$

where

$$\mathcal{L}_{\text{reg}}(\mathbf{W}_t) = \sum_l \Big[\lambda_0 \underbrace{(\|\mathbf{U}_i^{(k_l)\text{free}}\| + \|\mathbf{V}_i^{(k_l)\text{free}}\|)}_{L_1-\text{regularization}} + \lambda_1 \underbrace{(\|\mathbf{R}_i^{\text{free}}\|_2^2 + \|\mathbf{S}_i^{\text{free}}\|_2^2 + \|\mathbf{U}_i^{(k_l)\text{free}}\|_2^2 + \|\mathbf{V}_i^{(k_l)\text{free}}\|_2^2)}_{L_2-\text{regularization}} \Big] \tag{2}$$

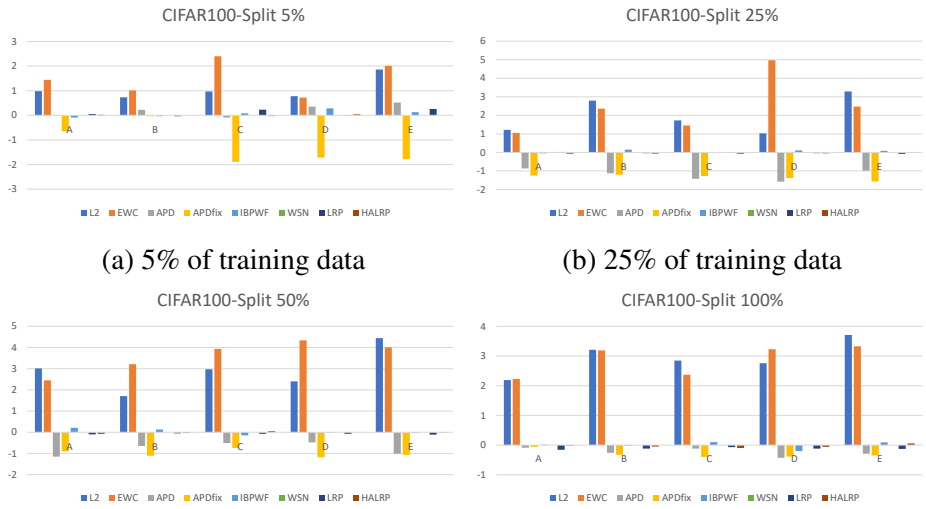
The fine-tuning objective mainly consists of three parts, the general cross-entropy loss on the new task $\mathcal{T}_t$, a L_1 regularization term on $\|\mathbf{U}_i^{(k_l)\text{free}}\| + \|\mathbf{V}_i^{(k_l)\text{free}}\|$, and the L_2 regularization terms on $\|\mathbf{R}_i^{\text{free}}\|_2^2 + \|\mathbf{S}_i^{\text{free}}\|_2^2$ and $\|\mathbf{U}_i^{(k_l)\text{free}}\|_2^2 + \|\mathbf{V}_i^{(k_l)\text{free}}\|_2^2$, respectively.

As mentioned in the thesis, we apply the L_1 regularization on $\mathbf{U}_i^{(k_l)\text{free}}$ and $\mathbf{V}_i^{(k_l)\text{free}}$. Besides, as discussed in Section 3.5 of the thesis, we also apply to prune $\mathbf{U}_i^{\text{free}}$ and $\mathbf{V}_i^{(k_l)\text{free}}$, which further encouraged the sparsity.

Our method keeps $\mathbf{W}^{\text{base}}$ as unchanged for new tasks for knowledge transfer, while the $\mathbf{R}_i^{\text{free}}$, $\mathbf{S}_i^{\text{free}}$, $\mathbf{U}_i^{(k)\text{free}}$ and $\mathbf{V}_i^{(k)\text{free}}$ are left as task-adaptive parameters. Since $\mathbf{R}_i^{\text{free}}$ and $\mathbf{S}_i^{\text{free}}$ are diagonal matrices, plus the fact that $\mathbf{U}_i^{(k)\text{free}}$ and $\mathbf{V}_i^{(k)\text{free}}$ are sparse, thus the model only needs to learn a small number of parameters. Unlike some regularization methods (*e.g.* EWC [7]), which constrain the gradient update and require to re-train a lot of parameters, our method can update the model more efficiently. In addition, the empirical results reported in the thesis show that our method can also achieve better performance with less time and memory cost.

	P-MNIST		
	Acc.↑	MOPD↓	AOPD↓
IBWPF	78.12 ± 0.83	12.69	6.65
HALRP	98.10 ± 0.03	0.47	0.24
	Five-dataset		
	Acc.↑	MOPD↓	AOPD↓
IBWPF	84.62 ± 0.36	5.06	1.72
HALRP	88.81 ± 0.31	4.28	1.31

Table D.1: Comparison with low-rank factorization method IBWPF



(a) 5% of training data

(b) 25% of training data

(c) 50% of training data

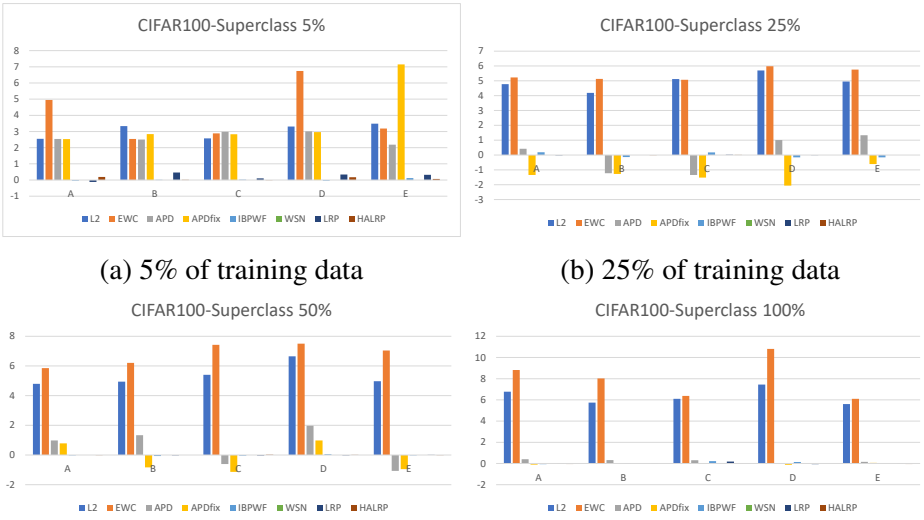
(d) 100% of training data

Figure D.1: Forgetting comparison on CIFAR100-Split with different task orders (A-E) under different amounts of training data.

D Additional Experimental Results

D.1 Comparing with other low-rank methods

Our work also shares some similarities with the low-rank-decomposition-based method IBPWF [12]. As discussed in [3] and in our thesis (Section 1.1), this method requires larger ranks to accept higher accuracy. Furthermore, as pointed out by [59], IBPWF leverages Bayesian non-parametric to let the data dictate expansion, but the benchmarks considered in Bayesian methods have been limited to smaller datasets, like MNIST and CIFAR-10.



(a) 5% of training data

(b) 25% of training data

(c) 50% of training data

(d) 100% of training data

Figure D.2: Forgetting on CIFAR100-SuperClass with different task orders (A-E) under different amounts of training data.

In addition to the experimental comparisons with IBPWF reported in the thesis, we further report more results in Table D.1. We can observe that our method outperforms IBPWF with a large gap in terms of Accuracy and MOPD↓ & AOPD↓. Furthermore, we observe that IBPWF cannot perform well on P-MNIST, which is 10× the number of data of MNIST. The gap is consistent with observations in [59].

D.2 Performance on Alleviate Forgetting on Different Task Orders

In the main thesis, we report the forgetting performance on the average of five task orders (A-E) on CIFAR100 Splits and SuperClass. In this part, we provide the performance on forgetting under each task order in Fig. D.1 and Fig. D.2, under different amounts (i.e., 5% ∼ 100%) of training data.

E Experimental Details

E.1 Dataset Preparation

CIFAR100 Splits/SuperClass We used the CIFAR100 Splits/SuperClass dataset following the evaluation protocol of [2], and follow the task order definition of [2] to test the algorithms with five different task orders (A-E).

For the CIFAR100 Split, the task orders are defined as:

- Order A: [0,1,2,3,4,5,6,7,8,9]
- Order B: [1, 7, 4, 5, 2, 0, 8, 6, 9, 3]
- Order C: [7, 0, 5, 1, 8, 4, 3, 6, 2, 9]
- Order D: [5, 8, 2, 9, 0, 4, 3, 7, 6, 1]
- Order E: [2, 9, 5, 4, 8, 0, 6, 1, 3, 7]

For the CIFAR100 SuperClass, the task orders are:

- Order A: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
- Order B: [15, 12, 5, 9, 7, 16, 18, 17, 1, 0, 3, 8, 11, 14, 10, 6, 2, 4, 13, 19]
- Order C: [17, 1, 19, 18, 12, 7, 6, 0, 11, 15, 10, 5, 13, 3, 9, 16, 4, 14, 2, 8]
- Order D: [11, 9, 6, 5, 12, 4, 0, 10, 13, 7, 14, 3, 15, 16, 8, 1, 2, 19, 18, 17]
- Order E: [6, 14, 0, 11, 12, 17, 13, 4, 9, 1, 7, 19, 8, 10, 3, 15, 18, 5, 2, 16]

Furthermore, as discussed in the thesis, we demonstrate the performance when handling limited training data. In this regard, we randomly select 5%, 25%, 50% training data from each task and report the corresponding accuracies.

P-MNIST We follow [1] to evaluate the algorithms' performance on the P-MNIST dataset.

Each task of P-MNIST is a random permutation of the original MNIST pixel. We follow [1, 48] to generate the train/val/test splits and to create 10 sequential tasks using different permutations, and each task has 10 classes. We randomly generate five different task orders with five different seeds.

- seed 0: [6, 1, 9, 2, 7, 5, 8, 0, 3, 4]
- seed 1: [2, 9, 6, 4, 0, 3, 1, 7, 8, 5]
- seed 2: [4, 1, 5, 0, 7, 2, 3, 6, 9, 8]
- seed 3: [5, 4, 1, 2, 9, 6, 7, 0, 3, 8]
- seed 4: [3, 8, 4, 9, 2, 6, 0, 1, 5, 7]

Five dataset It uses a sequence of 5 different benchmarks including CIFAR10 [49], MNIST [47], notMNIST [50], FashionMNIST [51] and SVHN [52]. Each benchmark contains 10 classes. We follow [1] to generate the train/val/test splits and to create 10 sequential tasks using different permutations, and each task has 10 classes. We randomly generate five different task orders with five different seeds.

- seed 0: [2, 0, 1, 3, 4]
- seed 1: [1, 0, 4, 2, 3]
- seed 2: [2, 4, 1, 3, 0]
- seed 3: [3, 4, 1, 0, 2]
- seed 4: [0, 3, 1, 4, 2]

Omniglot-rotation We split this dataset [53] into 100 12-way classification tasks. We follow the train/val/test split of [1]. The five task order adopted in our experiments are:

- seed 0: [26, 86, 2, 55, 75, 93, 16, 73, 54, 95, 53, 92, 78, 13, 7, 30, 22, 24, 33, 8, 43, 62, 3, 71, 45, 48, 6, 99, 82, 76, 60, 80, 90, 68, 51, 27, 18, 56, 63, 74, 1, 61, 42, 41, 4,

15, 17, 40, 38, 5, 91, 59, 0, 34, 28, 50, 11, 35, 23, 52, 10, 31, 66, 57, 79, 85, 32, 84, 14, 89, 19, 29, 49, 97, 98, 69, 20, 94, 72, 77, 25, 37, 81, 46, 39, 65, 58, 12, 88, 70, 87, 36, 21, 83, 9, 96, 67, 64, 47, 44]

- seed 1: [80, 84, 33, 81, 93, 17, 36, 82, 69, 65, 92, 39, 56, 52, 51, 32, 31, 44, 78, 10, 2, 73, 97, 62, 19, 35, 94, 27, 46, 38, 67, 99, 54, 95, 88, 40, 48, 59, 23, 34, 86, 53, 77, 15, 83, 41, 45, 91, 26, 98, 43, 55, 24, 4, 58, 49, 21, 87, 3, 74, 30, 66, 70, 42, 47, 89, 8, 60, 0, 90, 57, 22, 61, 63, 7, 96, 13, 68, 85, 14, 29, 28, 11, 18, 20, 50, 25, 6, 71, 76, 1, 16, 64, 79, 5, 75, 9, 72, 12, 37]

- seed 2: [83, 30, 56, 24, 16, 23, 2, 27, 28, 13, 99, 92, 76, 14, 0, 21, 3, 29, 61, 79, 35, 11, 84, 44, 73, 5, 25, 77, 74, 62, 65, 1, 18, 48, 36, 78, 6, 89, 91, 10, 12, 53, 87, 54, 95, 32, 19, 26, 60, 55, 9, 96, 17, 59, 57, 41, 64, 45, 97, 8, 71, 94, 90, 98, 86, 80, 50, 52, 66, 88, 70, 46, 68, 69, 81, 58, 33, 38, 51, 42, 4, 67, 39, 37, 20, 31, 63, 47, 85, 93, 49, 34, 7, 75, 82, 43, 22, 72, 15, 40]

- seed 3: [93, 67, 6, 64, 96, 83, 98, 42, 25, 15, 77, 9, 71, 97, 34, 75, 82, 23, 59, 45, 73, 12, 8, 4, 79, 86, 17, 65, 47, 50, 30, 5, 13, 31, 88, 11, 58, 85, 32, 40, 16, 27, 35, 36, 92, 90, 78, 76, 68, 46, 53, 70, 80, 61, 18, 91, 57, 95, 54, 55, 28, 52, 84, 89, 49, 87, 37, 48, 33, 43, 7, 62, 99, 29, 69, 51, 1, 60, 63, 2, 66, 22, 81, 26, 14, 39, 44, 20, 38, 94, 10, 41, 74, 19, 21, 0, 72, 56, 3, 24]

- seed 4: [20, 10, 96, 16, 63, 24, 53, 97, 41, 47, 43, 2, 95, 26, 13, 37, 14, 29, 35, 54, 80, 4, 81, 76, 85, 60, 5, 70, 71, 19, 65, 62, 27, 75, 61, 78, 18, 88, 7, 39, 6, 77, 11, 59, 22, 94, 23, 12, 92, 25, 83, 48, 17, 68, 31, 34, 15, 51, 86, 82, 28, 64, 67, 33, 45, 42, 40, 32, 91, 74, 49, 8, 30, 99, 66, 56, 84, 73, 79, 21, 89, 0, 3, 52, 38, 44, 93, 36, 57, 90, 98, 58, 9, 50, 72, 87, 1, 69, 55, 46]

TinyImageNet This dataset contains 200 classes. In our experiments, we adopted two split settings: 20 split and 40 split.

Each task in the 20-split setting consists of 10 classes. We adopted five random tasks orders as follows:

- seed 0: [10, 18, 16, 14, 0, 17, 11, 2, 3, 9, 5, 7, 4, 19, 6, 15, 8, 1, 13, 12]
- seed 1: [11, 5, 17, 19, 9, 0, 16, 1, 15, 6, 10, 13, 14, 12, 7, 3, 8, 2, 18, 4]
- seed 2: [7, 6, 17, 8, 19, 15, 13, 0, 3, 9, 14, 4, 10, 12, 16, 5, 11, 18, 2, 1]
- seed 3: [8, 3, 6, 5, 15, 16, 2, 12, 0, 1, 13, 10, 19, 9, 14, 11, 4, 17, 18, 7]
- seed 4: [17, 19, 10, 14, 5, 18, 16, 11, 4, 8, 6, 0, 13, 1, 2, 15, 12, 3, 9, 7]

Each task in the 40-split setting consists of 5 classes. We adopted five random task orders as follows:

- seed 0: [22, 20, 25, 4, 10, 15, 28, 11, 18, 29, 27, 35, 37, 2, 39, 30, 34, 16, 36, 8, 13, 5, 17, 14, 33, 7, 32, 1, 26, 12, 31, 24, 6, 23, 21, 19, 9, 38, 3, 0]
- seed 1: [2, 31, 3, 21, 27, 29, 22, 39, 19, 26, 32, 17, 30, 36, 33, 28, 4, 14, 10, 35, 23, 24, 34, 20, 18, 25, 6, 13, 7, 38, 1, 16, 0, 15, 5, 11, 9, 8, 12, 37]
- seed 2: [27, 9, 14, 0, 2, 30, 13, 36, 17, 37, 38, 29, 24, 12, 16, 1, 33, 23, 25, 19, 32, 10, 4, 6, 3, 34, 5, 28, 20, 26, 39, 21, 35, 31, 7, 11, 18, 22, 8, 15]
- seed 3: [29, 16, 9, 27, 4, 18, 28, 38, 15, 26, 25, 11, 30, 32, 13, 34, 39, 37, 5, 1, 31, 2, 22, 17, 14, 7, 12, 20, 36, 6, 23, 35, 33, 10, 19, 21, 0, 8, 3, 24]
- seed 4: [28, 39, 4, 15, 26, 20, 31, 7, 16, 11, 19, 33, 12, 18, 38, 13, 10, 22, 32, 25, 17, 36, 29, 14, 2, 24, 27, 6, 35, 34, 21, 37, 0, 3, 30, 9, 8, 23, 1, 5]

E.2 Model Architecture

In the experimental evaluations, we implement various backbone architectures to demonstrate our perturbation method for different deep models. We introduce the model architectures used in the thesis:

Dataset	CIFAR100 Split	CIFAR100 Super	PMNIST	Five dataset		Omniglot	TinyImageNet 40-split		TinyImageNet 20-split	
Network	LeNet	LeNet	LeNet	AlexNet	ResNet18	extended LeNet	AlexNet	ResNet18	AlexNet	ResNet18
n	20	20	12	12	12	20	50	50	50	50
n_w	1	1	1	1	3	1	20	25	25	25
α	0.9	0.9	0.9	0.9	0.95	0.99	0.9	0.9	0.9	0.9
LR	1e-3	1e-3	1e-3	1e-3	1e-3	5e-3	1e-3	5e-4	1e-3	5e-4
λ_0	1e-4	1e-4	1e-6	1e-6	1e-6	9e-5	5e-4	5e-4	1e-5	5e-4
λ_1	1e-4	1e-4	1e-3	1e-4	1e-4	1e-4	1e-4	1e-4	5e-4	1e-4
Bcsz	128	128	128	128	128	16	32	32	32	32

Table E.1: Hyperparameters for the experiments. n: total epoch. n_w: warm-up epochs for a new task. LR: Learning rate. λ_0, λ_1: coefficients for the regularization terms as discussed in Appendix C. Bcsz: training batch size.

LeNet We implement two kinds of LeNet models: **1):** For CIFAR100 Splits/SuperClass and P-MNIST, we implement the general LeNet model with neurons 20-20-50-800-500. **2):** For Omniglot-Rotation, we follow [1, 2] to implement the enlarged LeNet model with neurons 64-128-2500-1500.

AlexNet For the experiments on Five-dataset and TinyImageNet, we implement AlexNet model by following [1, 5].

ResNet-18 We adopted the reduced ResNet-18 model (i.e., reduce half of the filters in each convolutional layer from the standard Resnet18) on the Five-dataset and TinyImageNet by following [1].

E.3 Training Hyperparameters

We reimplement the baselines by rigorously following the official code release or publicly accessible implementations and tested our proposed algorithm with a unified test-bed with the same hyperparameters to get fair comparison results. The training details for our experiments are illustrated in Table E.1. The hyperparameters are selected via grid search. We also provide descriptions of the hyperparameters in the source code.

F Copyright Permission

Hessian Aware Low-Rank Perturbation for Order-Robust Continual Learning

Author: Jiaqi Li

Publication: IEEE Transactions on Knowledge and Data Engineering

Publisher: IEEE

Date: November 2024

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Curriculum Vitae

Name:	Jiaqi Li
Post-Secondary Education and	University of Western Ontario London, ON, Canada 2021 - 2024, M.Sc of Computer Science Beihang University Beijing, China 2016 - 2019, M.Eng of Computer Science Beihang University Beijing, China 2012 - 2016, B.Sc of Math and Applied Math
Related Work Experience:	Teaching Assistant The University of Western Ontario 2021 - 2024

Publications:

Jiaqi Li, Yuanhao Lai, Rui Wang, Changjian Shui, Sabyasachi Sahoo, Charles X. Ling, Shichun Yang, Boyu Wang, Christian Gagné, Fan Zhou. (2024). *Hessian Aware Low-Rank Perturbation for Order-Robust Continual Learning.* [Accepted]. In IEEE Transactions on Knowledge and Data Engineering. doi: 10.1109/TKDE.2024.3419449.