Flattening Sharpness for Dynamic Gradient Projection Memory Benefits Continual Learning

Danruo Deng[1], **Guangyong Chen**[2,*], **Jianye Hao**[3,4], **Qiong Wang**[2], **Pheng-Ann Heng**[1,2]
[1]The Chinese University of Hong Kong, [2]Guangdong-Hong Kong-Macao Joint Laboratory of Human-Machine Intelligence-Synergy Systems, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, [3]College of Intelligence and Computing, Tianjin University, [4]Huawei Noah's Ark Lab
{drdeng,pheng}@cse.cuhk.edu.hk, {gy.chen, wangqiong}@siat.ac.cn, jianye.hao@tju.edu.cn

Abstract

The backpropagation networks are notably susceptible to catastrophic forgetting, where networks tend to forget previously learned skills upon learning new ones. To address such the 'sensitivity-stability' dilemma, most previous efforts have been contributed to minimizing the empirical risk with different parameter regularization terms and episodic memory, but rarely exploring the usages of the *weight loss landscape*. In this paper, we investigate the relationship between the weight loss landscape and sensitivity-stability in the continual learning scenario, based on which, we propose a novel method, *Flattening Sharpness for Dynamic Gradient Projection Memory (FS-DGPM)*. In particular, we introduce a soft weight to represent the importance of each basis representing past tasks in GPM, which can be adaptively learned during the learning process, so that less important bases can be dynamically released to improve the sensitivity of new skill learning. We further introduce *Flattening Sharpness (FS)* to reduce the generalization gap by explicitly regulating the flatness of the weight loss landscape of all seen tasks. As demonstrated empirically, our proposed method consistently outperforms baselines with the superior ability to learn new skills while alleviating forgetting effectively.[2].

1 Introduction

Humans have the ability to continually learn new knowledge without forgetting their previously learned ones through mediating a rich set of neurocognitive mechanisms [41, 15, 39]. This ability, often known as continual learning or lifelong learning [29], is crucial for computational systems, such as deep neural networks (DNNs), which are required to sequentially learn and deal with multiple tasks when implemented in the dynamically changing environment. Continual learning remains a long-standing challenge for DNNs since these networks are typically trained with stationary training batches by stochastic gradient descent methods [19], which generally leads to an abrupt performance decrease on previously learned tasks as new tasks are learned. To address such *catastrophic forgetting*, we can brutally retrain an oracle network on the entire dataset containing all tasks to capture dynamic changes in the data distribution, but this methodology is obviously too inefficient to hinder the learning of novel data in real time.

During the last few years, lots of research efforts have been devoted to improving the *stability* of DNNs on old tasks while keeping *sensitive* to new information. The first intuitive idea is to introduce

*Corresponding author: Guangyong Chen. <gy.chen@siat.ac.cn>
[2]The code is available at: https://github.com/danruod/FS-DGPM

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

an independent branch for each new task while freezing the old task parameters to preserve the old knowledge [33, 45, 43, 25, 35, 22]. However, in this way, the network will inevitably become redundant as the task number continually increases. As presented in the neurocognitive works [41, 15], the reactivation of neuronal activity patterns, representing old memories, plays an important role in the continual learning of humans [39]. Thus, forgetness can be effectively mitigated by training a single network for new tasks by considering diverse information stored in the memory, including the original training samples of old tasks [31, 5, 13], the gradients induced from old tasks [9] and the feature subspace representing old tasks [34]. However, their continual learning performance is still limited because DNNs can easily overfit the limited information stored in the small-size memory.

The overfitting problem of DNNs is often attributed to the complex loss landscape containing multiple local optima, and the sharpness of the loss landscape has been widely used to characterize the generalization gap in standard training scenarios from both theoretical and empirical perspectives [27, 21, 42, 10, 6]. While this characterization has inspired new approaches for model training with better generalization, practical algorithms that especially seek out flatter minima to effectively address the 'sensitivity-stability' dilemma for continual learning have thus far been elusive. In this paper, our first contribution is to characterize the weight loss landscape for the continual learning scenario and identify that a flatter loss landscape with lower loss value often leads to better continual learning performance, as shown in Figure 1 and Figure 3.

Further, based on our characterization of the weight loss landscape, we find that the recently proposed Gradient Projection Memory (GPM) method [34] maintains the lowest loss value on old tasks among the previously proposed methods by taking gradient steps orthogonal to the subspace representing old tasks. However, its loss landscape on newly learned tasks is the sharpest due to the lack of sufficient subspace left for new task learning. To improve the network's sensitivity, our second contribution is to predict the importance of bases spanning the subspace for old tasks, so that less important bases can be dynamically released. In particular, we introduce a soft weight to indicate the bases importance, which can be dynamically adjusted by combining the *Flattening Sharpness (FS)* to minimize the loss value and loss sharpness simultaneously. Intuitively, a basis will be regarded as important for preserving old knowledge if the gradients induced by new tasks and old ones are aligned in the opposite direction on that basis. As demonstrated through extensive experiments, our proposed method can consistently outperform the state-of-the-art methods [17, 30, 24, 31, 5, 13, 34] by a notable margin across a range of widely used benchmark datasets.

2 Related Work

In this section, we briefly survey the representative works of continual learning by highlighting their contributions. To simplify our presentation, this section is organized by dividing the representative works into three categories, parameter isolation, regularization-based, memory-based methods.

Parameter isolation methods address forgetting by assigning a different subset of network parameters to each task. Without restrictions on network architecture, new neurons or layers or modules can be added for new tasks, while the previous task parameters can be frozen or copied to preserve old knowledge. For instance, Progressive Neural Network (PGN) [33] freezes the parameters trained with previous knowledge while expands the architecture by allocating new sub-networks with fixed capacity for new tasks. Dynamically Expandable Networks (DEN) [45] selectively retrains or expands network capacity by splitting/duplicating important units on new tasks. Reinforced Continual Learning (RCL) [43] uses reinforcement learning strategy to adaptively expand the network of each layer, while [22] use neural architecture search to find optimal network structures for each sequential task. Alternatively, with the architecture remaining static, a fixed part is allocated to each task. During the training of a new task, previous task parts are masked out to prevent interference. The mask sets are imposed at parameter level [25], or unit level [35]. PackNet [25] uses iterative pruning to fully restrict gradient updates on important weights via a binary mask, whereas HAT [35] limits the update of important units recognized by the hard attention mask through gradient backpropagation.

Regularization-based methods introduce an additional regularization term in the loss function to consolidate previous knowledge without using replay. This involves using knowledge distillation [23, 14] or penalizing changes in weights deemed important for previous tasks [17, 46, 28, 2, 3] to reduce forgetting. There are many ways to measure the importance. Elastic Weight Consolidation (EWC) [17] identifies important weights based on the diagonal values of Fisher information matrix

after training, while Synaptic Intelligence (SI) [46] calculates them online and over the entire learning trajectory in parameter space. Memory Aware Synapses (MAS) [2] estimates importance weights in an unsupervised manner, while Variational Continual Learning (VCL) [28] introduces a variational framework that spawned some Bayesian-based works [32, 8, 1, 7]. For example, [32] recursively uses a Gaussian Laplace approximation of the Hessian to approximate the posterior after every task, [8] adjusts the learning rate according to the uncertainty defined by the probability distribution of the network weights. [1] introduces an interpretation of node-wise uncertainty on the Kullback-Leibler (KL) divergence term of the variational lower bound for Gaussian mean-field approximation.

Our method mainly follows **memory-based** methods, which mitigate forgetting based on information extracted from old tasks or based on a generative model to generate pseudo samples. For example, iCaRL [30] selects and stores samples closest to the feature mean of each class. ER [5, 31] suggests reservoir sampling under the limited and fixed budget for replay buffer. Deep Generative Replay (DGR) [36] trains a deep generative model in the Generative Adversarial Network (GAN) framework [11] to simulate past data. These previous task samples are mainly reused as model inputs for replay in the above methods. However, replay might be prone to overfitting the subset of stored samples. Alternatively, samples stored in memory can also be used to constrain the optimization of the new task loss to prevent previous task interference, thereby leaving more leeway for backward and forward transfer. Gradient Episodic Memory (GEM) [24] projects the estimated gradients in the feasible region, which is outlined by previous task gradients calculated from the episodic memory samples. Averaged-GEM (A-GEM) [4] relaxes the projection to a direction that is estimated from samples randomly selected from memory. [12] proposes a unified view of episodic memory-based continual learning methods, including GEM and A-GEM, and improves performance over these methods by using a loss-balancing update scheme. A few other works have utilized gradient information to protect previous knowledge. [31, 13] adopt optimization-based meta-learning to enforce gradient alignment between samples from different tasks. GPM [34] minimizes interference between sequential tasks by ensuring that gradient updates only occur in directions orthogonal to the input of previous tasks.

3 The Weight Loss Landscape of Continual Learning

In this section, we first introduce our formulation of continual learning, and then characterize the weight loss landscape for the continual learning scenario from stability and sensitivity. Finally, some insights combining the weight loss landscape and continual learning are provided.

3.1 Problem Formulation

Throughout the paper, we denote scalars as a, vectors as $\boldsymbol{a}$, matrices as $\boldsymbol{A}$, and sets as $\mathcal{A}$. We consider a supervised learning setup where T tasks are sequentially learned from their training data. Each task has an identical task descriptor, $\tau \in \{1, 2, \ldots, T\}$, with its dataset $\mathcal{D}_\tau = \{\boldsymbol{x}_{i,\tau}, y_{i,\tau}\}_{i=1}^{n_\tau}$ containing n_τ samples randomly generated from a latent distribution $\mathscr{D}_\tau$. At any time-step during the learning process, we minimize the empirical risk of the model on all t tasks seen so far, with just limited size of memory $\mathcal{M}$ to summarize the training data of previous tasks $\{\mathcal{D}_\tau\}_{\tau=1}^{t-1}$. To simplify the notation, we denote $L_{\mathcal{A}}(\boldsymbol{w}) = \frac{1}{|\mathcal{A}|}\sum_{(\boldsymbol{x},y)\in\mathcal{A}}\left[\ell\left(f_{\boldsymbol{w}}\left(\boldsymbol{x}\right), y\right)\right]$ as the average empirical loss for the set $\mathcal{A}$, where $\ell(\cdot,\cdot)$ is an arbitrary loss function (*e.g.* the cross-entropy (CE) loss), $|\mathcal{A}|$ is the sample size of the set $\mathcal{A}$, and $f_{\boldsymbol{w}}$ is the DNN with weight $\boldsymbol{w}$. Our final goal is to find an optimal parameter $\boldsymbol{w}$, which minimizes the overall risk $\sum_{\tau=1}^{T} L_{\mathscr{D}_\tau}(\boldsymbol{w})$ for all tasks.

3.2 Connection of Weight Loss Landscape and Continual Learning

After learning a new task, we visualize the weight loss landscape of each task seen so far by plotting changes in its training loss when moving the weights $\boldsymbol{w}$ in a random direction $\boldsymbol{d}$ with magnitude α following [21]:

$$g_t(\alpha) = L_{\mathcal{D}_t}(\boldsymbol{w} + \alpha\boldsymbol{d}) = \frac{1}{|\mathcal{D}_t|}\sum_{(\boldsymbol{x},y)\in\mathcal{D}_t}\ell\left(f_{\boldsymbol{w}+\alpha\boldsymbol{d}}\left(\boldsymbol{x}\right), y\right),$$

where $\mathcal{D}_t$ is the training set for the t-th task previously learned. To eliminate the scaling invariance of DNNs, $\boldsymbol{d}$ is sampled from a Gaussian distribution and further normalized by $\boldsymbol{d}_{l,j} \leftarrow \frac{\boldsymbol{d}_{l,j}}{\|\boldsymbol{d}_{l,j}\|_F}\|\boldsymbol{w}_{l,j}\|_F$,

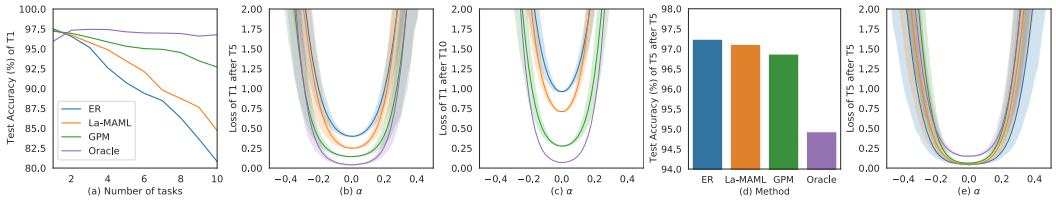

Figure 1: The connection between the weight loss landscape and continual learning is investigated on four methods. (a)-(c) shows the stability of the first task. (a) is the test accuracy change curve of the first task; (b) and (c) are the weight loss landscape of the first task after learning the fifth task and all ten tasks. (d)-(e) shows the sensitivity of the fifth task, which are the test accuracy and the weight loss landscape of the fifth task after just learning the fifth task. The shape of the weight loss landscape obtained using ten different random filter-normalized directions. ("Ti" is abbr. of the i-th task)

where $\boldsymbol{d}_{l,j}$ represents the j-th filter at l-th layer of $\boldsymbol{d}$, and $\|\cdot\|_F$ denotes the Frobenius norm. Compared with our visualization, [26] only consider one task and plot the loss landscape along the directions of its Hessian eigenvectors, which only reflects some of the relationship between forgetting and landscape. Considering $\boldsymbol{d}$ is randomly selected, we repeat the visualization 10 times with different $\boldsymbol{d}$.

We first study the *stability* of the network by plotting changes of the weight loss landscape for the first task after new task learning. In particular, We use the previously proposed ER [5], La-MAML [13], and GPM [34] to train a MLP network with two hidden layers on the Permuted MNIST (PMNIST) [20] dataset that contains 10 tasks. We also retrain the network on the entire dataset contain all passed tasks as an Oracle network. Early stopping is used to halt the training with up to 10 epochs for each task based on the validation loss as proposed in [35]. As shown in Figure 1(a), all three continual learning methods lose their stability as learning new tasks. It can be observed from Figure 1(b)-(c) that the weight loss landscape becomes sharper and loss value increases simultaneously, when the testing accuracy of the first task continually decreases. We further evaluate the *sensitivity* of the network by observing the performance of the fifth task just after it has just been learned. As shown in Figure 1(d)-(e), ER shows the best learning capability compared with other methods with the lowest loss value and the flattest loss landscape. Thus, based on these empirical findings, we assume that lower loss value with a flatter neighbor may lead to better continual learning performance.

3.3 A Case Study of Flattening Sharpness for Vanilla ER

In this part, we further validate our above assumption by Flattening Sharpness for vanilla ER (FS-ER). Compared with ER that looks for a solution $\boldsymbol{w}$ that jointly minimizes the training loss of current task data and memory data, FS-ER seeks out a solution with both low loss and flat neighbor by minimizing the maximal loss in the neighbor around the parameter value. The schematic of the Flattening Sharpness (FS) is shown in Figure 2. We introduce the adversarial weight perturbation (orange dashed line) to explicitly flatten the weight loss landscape via injecting the worst-case weight perturbation, which is calculated from the current task data and past task data sampled from the replay buffer (Refer to Appendix C.1 and the next section for more details). Figure 3 respectively shows the weight loss landscapes of the fifth task after just learning the fifth task and all tasks. We find that FS-ER successfully gets a solution with a lower loss value and flatter landscape, either after the fifth task has just been learned or all ten tasks have been learned. The average testing accuracy of all tasks using FS-ER is **90.44%**, significantly higher than ER (**86.16%**), which means that flattening sharpness does benefit continual learning.

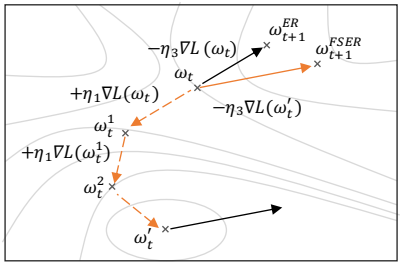

Figure 2: Schematic of FS-ER update. The dashed line and the solid line indicate the gradient ascent and descent, respectively. Orange denotes the actual update of the parameter $\boldsymbol{w}$.

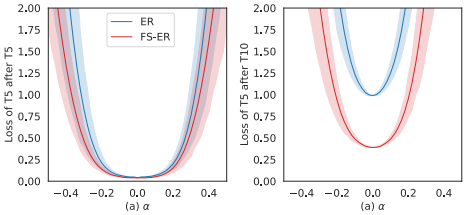

Figure 3: Landscape of the fifth task after just learning (a) the fifth task and (b) all ten tasks.

4 Flattening Sharpness for Dynamic Gradient Projection Memory

As shown in Figure 1, GPM achieves the highest testing accuracy on old tasks among all three practical continual learning methods, but shows less sensitivity to new task learning. To address this issue, we propose *Flattening Sharpness for Dynamic GPM (FS-DGPM)*, which dynamically adjusts the gradient subspace representing the past tasks to improve the sensitivity to the new task, while ensuring stability of the previous tasks. In particular, we let $\boldsymbol{M} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_k]$ denote the bases matrix that spans the gradient subspace of the previous task, $\boldsymbol{\Lambda} = \text{diag}[\lambda_1, \lambda_2, ..., \lambda_k]$ be the diagonal matrix with its i-th diagonal element $\lambda_i \in [0, 1]$ indicating the importance of each basis, and k is the number of bases.

4.1 Sharpness Evaluation

Comparing with the classical strategy that perturbs weight in the entire space [40, 16, 42, 10], we focus on characterizing the weight loss landscape on the new task with respect to the important subspace representing old task. The important subspace can be effectively calculated based on the examples sampled from replay buffer $\mathcal{M}$ after each task training. Then, we can find the worst case by maximizing the training loss of the network on the new task in this subspace. Formally, the sharpness of the loss landscape around the solution $\boldsymbol{w}$ in the old parameter space can be predicted as,

$$\max_{\boldsymbol{v} \in \mathcal{V}} \quad L_{\hat{\mathcal{D}}_t}\left(\boldsymbol{w} + \boldsymbol{v}\right), \tag{1}$$

where $\mathcal{V}$ denotes the subspace spanned by $\boldsymbol{M}$ and $\boldsymbol{\Lambda}$, and $\hat{\mathcal{D}}_t$ denotes the batch samples of the current t-th task. As shown in Eq. (1), the high value can be obtained when the network fails to learn the new task (*sensitivity*) and the new task learning seriously interferes with the past tasks learning (*stability*). Based on the gradient method, the adversarial weight perturbation $\boldsymbol{v}$ can be solved as,

$$\boldsymbol{v} \leftarrow \boldsymbol{v} + \eta_1 \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^T \left(\nabla_{(\boldsymbol{w}+\boldsymbol{v})} L_{\hat{\mathcal{D}}_t}(\boldsymbol{w} + \boldsymbol{v})\right), \tag{2}$$

where η_1 is the update step size. Note that $\boldsymbol{v}$ is initialized as $\mathbf{0}$ and layer-wise updated. As shown in Appendix E, two-step for $\boldsymbol{v}$ (default settings) are enough to get good improvements.

4.2 Dynamic Gradient Projection Memory

After obtaining the adversarial weight perturbation $\boldsymbol{v}$, we can further update the bases importance matrix $\boldsymbol{\Lambda} = \text{diag}[\lambda_1, \lambda_2, ..., \lambda_k]$ by jointly considering the current task batch $\hat{\mathcal{D}}_t$ and the batch $\hat{\mathcal{M}}$ sampled from the replay buffer $\mathcal{M}$ as following,

$$\lambda_i \leftarrow \lambda_i - \eta_2 \left(\nabla_{\lambda_i} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w} + \boldsymbol{v})\right), \tag{3}$$

where the sigmoid function is used at the end of gradient update to constrain the importance value λ_i between 0 and 1. In addition, the second term on the right side in Eq. (3) can be approximated by the first-order Taylor expansion as,

$$\nabla_{\lambda_i} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w} + \boldsymbol{v}) \approx \eta_1 \left(\nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t}\left(\boldsymbol{w}\right)\right)^T \boldsymbol{u}_i \boldsymbol{u}_i^T \left(\nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}\right)\right). \tag{4}$$

The above equation characterizes the relationship between the gradients induced by the current task and the old tasks based on the basis $\boldsymbol{u}_i$. As illustrated in the Figure 4, this equation implies that when the projections of two gradients on the basis $\boldsymbol{u}_i$ are aligned in the same direction, the gradient of λ_i will be positive, and when there is interference, the gradient will be negative. The positive (negative) gradient will decrease (increase) the importance λ_i, thereby releasing (tightening) the update limit of the new task on the corresponding basis $\boldsymbol{u}_i$. We provide the full derivation in the Appendix A.2. Note that the initial value of all importance is set to 1 and dynamically adjusted from the second task.

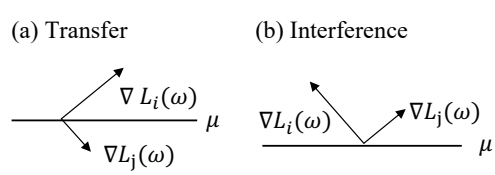

Figure 4: A depiction of transfer (a) and interference (b) in the basis $\boldsymbol{u}$ of gradient space.

4.3 Weight Updating

Finally, we update the model parameters by minimizing the worst performance of $f_{\boldsymbol{w}+\boldsymbol{v}}$ with respect to $\boldsymbol{w}$, while adjusting the update magnitude of $\boldsymbol{w}$ on each basis based on its importance to alleviate forgetting. More concretely, the parameter $\boldsymbol{w}$ will be updated to:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_3 \left(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^T \right) \nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w}+\boldsymbol{v}). \tag{5}$$

Note that the optimization is performed over the model parameters $\boldsymbol{w}$, whereas the objective is computed using perturbed model $f_{\boldsymbol{w}+\boldsymbol{v}}$. In addition, we update the replay buffer $\mathcal{M}$ with reservoir sampling as in [31], and then use Singular Value Decomposition (SVD) to recalculate $\boldsymbol{M}$ based on the sampling data in the replay buffer after learning one task following GPM [34]. Comparing with [34], we calculate the important bases in the entire gradient space and use them to replace the bases calculated last time. Besides that, our method degenerates to GPM when η_1 and η_2 are set to 0. The complete pseudo-code of FS-DGPM is outlined in the Algorithm 1.

Algorithm 1 FS-DGPM (Flattening Sharpness for Dynamic Gradient Projection Memory)

Input: Network weight $\boldsymbol{w}$, loss function ℓ, learning rate η_3, FS step size η_1, FS steps K, Soft weight step size η_2, batch size b.
Initializing $\mathcal{M} \leftarrow \{\}$, $\boldsymbol{M} \leftarrow \boldsymbol{I}, \boldsymbol{\Lambda} \leftarrow \boldsymbol{I}$
for $t = 1, 2, \cdots, T$ **do**
  for $ep = 1, 2, \cdots, num_{epochs}$ **do**
    for batch $\hat{\mathcal{D}}_t \overset{b}{\sim} \mathcal{D}_t$ **do**
      $\hat{\mathcal{M}} \overset{b}{\sim} \mathcal{M}$
      for $k = 1, \cdots, K$ **do**
        $\boldsymbol{v} \leftarrow \boldsymbol{v} + \eta_1 \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^T \left(\nabla_{(\boldsymbol{w}+\boldsymbol{v})} L_{\hat{\mathcal{D}}}(\boldsymbol{w}+\boldsymbol{v}) \right)$ ▷ Sharpness Evaluation
      end for
      if $t \geq 2$ **then**
        $\boldsymbol{\Lambda} \leftarrow \boldsymbol{\Lambda} - \eta_2 \left(\nabla_{\boldsymbol{\Lambda}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w}+\boldsymbol{v}) \right)$ ▷ Dynamic Gradient Projection Memory
      end if
      $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_3 \left(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^T \right) \nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w}+\boldsymbol{v})$ ▷ Weight updating
      Push $\hat{\mathcal{D}}_t$ to $\mathcal{M}$ with reservior sampling
    end for
  end for
  $\boldsymbol{M} \leftarrow$ UpdateGPM $(\mathcal{M})$ ▷ see Appendix Alg. 2
end for

4.4 Theoretical Understanding

We further provide a theoretical view on why landscape can characterize the continual learning performance and why our proposed FS-DGPM works. To simplify our explanation, we only consider two tasks, which contains the training sets $\mathcal{D}_1$ and $\mathcal{D}_2$ sampled from the distributions $\mathscr{D}_1$ and $\mathscr{D}_2$, respectively. Based on the previous works on PAC-Bayes bound [27, 42, 10], given a "prior" distribution P (a common assumption is zero mean, σ^2 variance Gaussian distribution) over the weights, the expected error of the classifier for the continual learning scenario can be bounded with probability at least $1-\delta$ over the draw of n training data:

$$\min_{\Delta \boldsymbol{w}} \mathbb{E}_{\boldsymbol{v}} \left[L_{\mathscr{D}_1 \cup \mathscr{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}) \right] \leq \min_{\Delta \boldsymbol{w} \in \mathcal{V}^C} \mathbb{E}_{\boldsymbol{v}} \left[L_{\mathscr{D}_1}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}) \right] + L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w})$$
$$+ \max_{\boldsymbol{v} \in \mathcal{V}} L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}) - L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}) + 4\sqrt{\frac{1}{n}\left(KL(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}||P) + \ln\frac{2n}{\delta} \right)}.$$

where $\Delta\boldsymbol{w}$ is the update based on the previously optimal solution $\boldsymbol{w}$ learned on the old task $\mathcal{D}_1$ when learning the new one $\mathcal{D}_2$, and $\boldsymbol{v}$ is often chosen as a zero mean spherical Gaussian perturbation with variance σ^2 in every direction. Let $\Delta\boldsymbol{w} \in \mathcal{V}^C$, then $\Delta\boldsymbol{w}$ lies in the complementary space of the important space representing the old task $\mathscr{D}_1$, so that $\mathbb{E}_{\boldsymbol{v}} \left[L_{\mathscr{D}_1}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}) \right]$ does not increase

too much compared with the previously minimized $\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1}(\boldsymbol{w}+\boldsymbol{v})\right]$. The second term denotes the empirical loss on the second task and the third term represents the sharpness of the weight loss landscape around the $\boldsymbol{w}+\Delta\boldsymbol{w}$. Since we have constrained $\Delta\boldsymbol{w} \in \mathcal{V}^C$, then it is natural to assume $\boldsymbol{v} \in \mathcal{V}$, so that $\Delta\boldsymbol{w}+\boldsymbol{v}$ will cover the full space. Thus, our FS-DGPM exactly optimizes the worst-case of the flatness of weight loss landscape to control the PAC-Bayes bound, which theoretically justifies both lower loss value and flatter landscape lead to better continual learning performance, and why our proposed FS-DGPM works.

5 Experiments

In this section, we conduct extensive experiments to compare the performance of our proposed FS-DGPM model with the state-of-the-art methods on widely used continual learning benchmark datasets. Additional results and more details about the datasets, experiment setup, baselines, and model architectures are presented in the Appendix D and E.

5.1 Experimental Setup

Datasets: We evaluate our algorithm on four image classification datasets: **Permuted MNIST** (PMNIST) [20], **CIFAR-100 Split** [18], **CIFAR-100 Superclass** [44] and **TinyImageNet** [37]. The PMNIST dataset is a variant of the MNIST dataset, in which each task applies a fixed random pixel permutation to the original dataset. The PMNIST benchmark dataset consists of 20 tasks, and each contains only 1000 samples from 10 different classes [13]. The CIFAR-100 Split is constructed by randomly dividing 100 classes of CIFAR-100 into 10 tasks with 10 classes per task. The CIFAR-100 Superclass is divided into 20 tasks according to the 20 superclasses of the CIFAR-100 dataset, and each superclass contains 5 different but semantically related classes. Whereas, TinyImageNet is constructed by splitting 200 classes into 40 5-way classification tasks. In our experiments, we do not use any data augmentation. The dataset statistics are given in Appendix D.1.

Network Architecture: For PMNIST, we use a fully connected network with two hidden layers of 100 units each following [24]. For experiments of CIFAR-100 Split and CIFAR-100 Superclass, we use a 5-layer AlexNet and LeNet architecture similar to [34] respectively. For TinyImageNet, we use the same network architecture as [13], which consists of 4 conv layers and 3 fully connected layers. In PMNIST, all tasks share the final classifier layer, while other experiments use a multi-head incremental setting, that is, each task has a separate classifier. Refer to Appendix D.2 for more details.

Baselines: We compare our method against multiple methods described below. (1) **EWC** [17], a regularization-based method that uses the diagonal of Fisher information to identify important weights; (2) **ICARL** [30], a memory-based method that uses knowledge-distillation and episodic memory to reduce forgetting; (3) **GEM** [24], another memory-based method that uses the gradient of episodic memory to constrain optimization to prevent forgetting; (4) **ER** [5], a simple and competitive method based on reservoir sampling; (5) **La-MAML** [13] and (6) **GPM** [34] are memory-based methods inspired by optimization-based meta-learning and based on gradient orthogonal constraints, respectively; (7) **Multitask** is an oracle baseline that all tasks are learned jointly using the entire dataset at once in a single network. Multitask is not a continual learning strategy but serves as an upper bound on average test accuracy on all tasks.

Training Details: All baselines and our method use stochastic gradient descent (SGD) for training. For each task in PMNIST and TinyImageNet, we train the network in 1 and 10 epochs, respectively, with the batch size as 10. These experimental settings are the same as La-MAML [13], so that we directly compare with their reported results. In the CIFAR-100 Split and CIFAR-100 Superclass experiments, we use the early termination strategy to train up to 50 epochs for each task, which is based on the validation loss as proposed in [35]. For both datasets, the batch size is set to 64. The replay buffer size of PMNIST, CIFAR-100 Split, CIFAR-100 Superclass, and TinyImageNet are 200, 1000, 1000, and 400, respectively. Details about the experimental setting and the hyperparameters considered for each baseline are provided in Appendix D.5 and D.6.

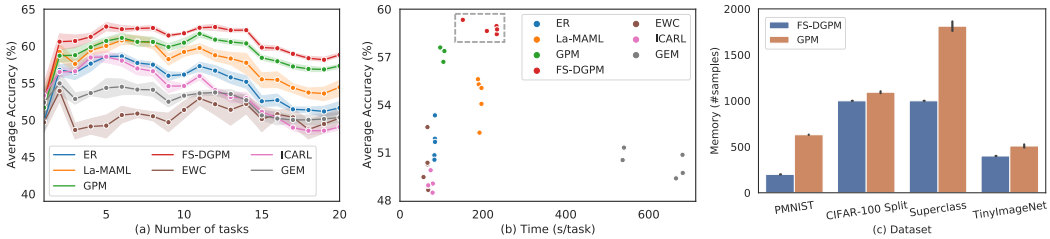

Figure 5: (a) Average accuracy as a function of the number of tasks trained on 20-Split CIFAR-100 Superclass. (b) Training time per task on 20-Split CIFAR-100 Superclass. (c) Memory usage on four datasets. ("Superclass" is abbr. of CIFAR-100 Superclass).

Table 1: Experimental results on 10-Split CIFAR-100, 20-Split CIFAR-100 Superclass and 40-Split TinyImageNet in 50 epochs. Each experiment is run with 5 seeds. † and * denotes results reported by [13] and [44] respectively.

	CIFAR-100 Split		**CIFAR-100 Superclass**		**TinyImageNet**	
Method	ACC(%)	BWT(%)	ACC(%)	BWT(%)	ACC(%)	BWT(%)
EWC	72.77 ± 0.45	-3.59 ± 0.55	50.26 ± 1.48	-7.87 ± 1.63	-	-
GEM	70.15 ± 0.34	-8.61 ± 0.42	50.35 ± 0.80	-9.50 ± 0.85	$50.57 \pm 0.61^\dagger$	$-20.50 \pm 0.10^\dagger$
ICARL	53.50 ± 0.81	-20.44 ± 0.82	49.05 ± 0.51	-11.24 ± 0.27	$54.77 \pm 0.32^\dagger$	$-3.93 \pm 0.55^\dagger$
ER	70.07 ± 0.35	-7.70 ± 0.59	51.64 ± 1.09	-7.86 ± 0.89	$48.32 \pm 1.51^\dagger$	$-19.86 \pm 0.70^\dagger$
La-MAML	71.37 ± 0.67	-5.39 ± 0.53	54.44 ± 1.36	-6.65 ± 0.85	$66.90 \pm 1.65^\dagger$	$-9.13 \pm 0.90^\dagger$
GPM	73.18 ± 0.52	$\mathbf{-1.17 \pm 0.27}$	57.33 ± 0.37	$\mathbf{-0.37 \pm 0.12}$	67.39 ± 0.47	$\mathbf{1.45 \pm 0.22}$
FS-DGPM	$\mathbf{74.33 \pm 0.31}$	-2.71 ± 0.17	$\mathbf{58.81 \pm 0.34}$	-2.97 ± 0.35	$\mathbf{70.41 \pm 1.30}$	-2.11 ± 0.84
Multitask	79.75 ± 0.38	-	$61.00 \pm 0.20^*$	-	$77.10 \pm 1.06^\dagger$	-

Metrics: We evaluate the continual learning performance by the *average accuracy* (ACC) and *backward transfer* (BWT) [24, 4, 5], formulated as following,

$$ACC = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}, \quad BWT = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i},$$

where T is the total number of learned sequential tasks, $R_{i,j}$ is the test classification accuracy of the model on j-th task after learning the last sample from i-th task. ACC is the average test classification accuracy of all tasks, bigger is better. BWT is the interference of new learning on the past knowledge. More specifically, negative BWT implies (catastrophic) forgetting whereas positive BWT indicates learning new task increases the performance on some preceding tasks.

5.2 Results and Discussion

PMNIST: We first evaluate our algorithm for 20 sequential PMNIST tasks with only 1000 samples per task in a single-head setting. From the results, as shown in Table 2, we see that our method (FS-DGPM) achieves the best average accuracy (**76.96%** $\pm$ **0.77**). Moreover, we achieve the least amount of forgetting except GPM, which is essentially a trade-off in accuracy to minimize forgetting. As shown in Figure 5(c), we only use about **31%** of the final memory size of GPM and achieve $\sim$ **2.5%** better accuracy.

Table 2: Experimental results (mean $\pm$ std in 5 runs) on PMNIST in single-epoch.

	PMNIST	
Method	ACC(%)	BWT(%)
EWC	62.25 ± 1.44	-15.22 ± 1.25
GEM	61.82 ± 0.85	-15.58 ± 1.17
ER	68.31 ± 0.56	-13.91 ± 0.67
La-MAML	75.98 ± 0.60	-10.21 ± 0.90
GPM	74.54 ± 0.36	$\mathbf{-7.17 \pm 0.51}$
FS-DGPM	$\mathbf{76.96 \pm 0.51}$	-7.45 ± 0.30
Multitask	86.54 ± 1.74	-

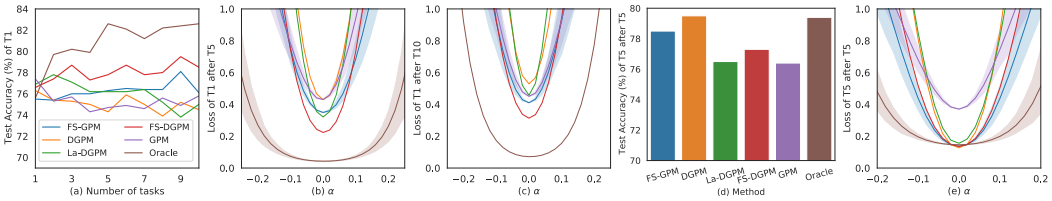

Figure 6: The ablation study implemented on CIFAR-100 Split with 10 tasks. (a)-(c) shows the stability of the first task. (a) is the test accuracy change curve of the first task; (b) and (c) are the weight loss landscape of the first task after learning the fifth task and all ten tasks. (d)-(e) shows the sensitivity of the fifth task, which are the test accuracy and the weight loss landscape of the fifth task after just learning the fifth task. The shape of the weight loss landscape obtained using ten different random filter-normalized directions. ("Ti" is abbr. of the i-th task)

CIFAR-100 and TinyImageNet: Next, we use a multi-head setting to evaluate our algorithm under the more challenging visual classification benchmarks. Table 1 reports all results of these experiments. We outperform all baselines on three datasets, with achieving the best average accuracy **74.33%**, **58.81%** and **70.41%**. In these experiments, we observe that GPM is a strong baseline with the least forgetting. At the same time, we highlight that our method achieves the highest accuracy on all datasets and the second-lowest forgetting after GPM. Figure 5(a) shows the process of performance changing with the number of tasks on the CIFAR-100 Superclass. We consistently see the superior performance of our method at any stage of model evolution. It is also worth emphasizing that although our method requires more time for training than GPM, it has lower memory usage and better test accuracy (See Figure 5(b)-(c)). As noted by [4] and [38], EWC performs poorly without multiple passes over the datasets, and GEM is not very effective under the single-head variants. These situations have also been observed in our experiments.

5.3 Ablation studies on FS-DGPM

We further investigate our model performance with an ablation study and summarize it in Table 3. We respectively ablate the effects of flattening sharpness and dynamically adjusting the soft weight for bases. We refer to them as DGPM and FS-GPM. We also construct an ablation referred to as La-DGPM (Look-ahead DGPM), where the adversarial weight perturbation $\boldsymbol{v}$ in Eq. (2) is changed to the direction of gradient descent. At the same time, we also change the sign in Eq. (3) to ensure that the soft weight of the basis is reduced when the gradients are aligned in the same direction. From the results, shown in Table 3, we observe that flattening sharpness does benefits GPM, with $\sim$ **1.0%** improvement over GPM on both datasets. We can further observe through Figure 6 that all landscapes of FS-DGPM have lower loss values and flatter neighbors than DGPM and La-DGPM on the CIFAR-100 Split experiments. In addition, we see that DGPM performs well in learning new tasks, but it also leads to forgetting previous tasks. This situation can be efficiently alleviated by flattening sharpness. Hence, FS is indeed a powerful method worthy of being widely adopted for continual learning scenarios.

Table 3: The ablation study results on CIFAR-100 Split and Superclass. Each experiment is run with 5 seeds.

	CIFAR-100 Split		**CIFAR-100 Superclass**	
Method	ACC(%)	BWT(%)	ACC(%)	BWT(%)
FS-DGPM	**74.33** $\pm$ 0.31	**-2.71** $\pm$ 0.17	**58.81** $\pm$ 0.34	-2.97 $\pm$ 0.35
La-DGPM	73.74 $\pm$ 0.61	-3.05 $\pm$ 0.73	58.18 $\pm$ 0.41	**-2.41** $\pm$ 0.39
FS-GPM	73.96 $\pm$ 0.44	-3.12 $\pm$ 0.43	58.61 $\pm$ 0.53	-2.79 $\pm$ 0.30
DGPM	73.78 $\pm$ 0.32	-3.67 $\pm$ 0.42	56.78 $\pm$ 0.49	-2.44 $\pm$ 0.40
GPM	73.18 $\pm$ 0.52	**-1.17** $\pm$ 0.27	57.33 $\pm$ 0.37	**-0.37** $\pm$ 0.12

6 Conclusion

In this paper, we explore the weight loss landscape to characterize the well-known 'sensitivity-stability' dilemma faced by continual learning algorithms, and find that lower loss value with flatter neighbor often leads to better continual learning performance. Based on this finding, we propose our FS-DGPM algorithm, which introduces a soft weight to represent the importance of each basis representing past tasks in GPM, so that less important bases can be dynamically released to improve

the sensitivity of new skill learning. Flattening Sharpness (FS) is also introduced here to reduce the generalization gap by explicitly regulating the flatness of the weight loss landscape of all tasks seen so far. The evaluation of various image classification tasks with different network architectures and the comparison with some state-of-the-art algorithms show the effectiveness of our method in achieving high classification performance while alleviating forgetting. Although our method theoretically and empirically demonstrates the advantages of introducing FS into continual learning, whether there exists a closer upper bound for the continual learning performance remains an unresolved problem and left for our future exploration. Although our method theoretically and empirically demonstrates the advantages of introducing bases soft weight and FS into continual learning, whether there exists a better dynamic adjustment and a closer upper bound for the continual learning performance remains an unresolved problem and left for our future exploration.

Acknowledgments

We would like to thank Dr. Yongzhe Deng and Zhuo Zhang for their helpful discussions, and anonymous reviewers for their valuable comments to improve this work. This work was supported by a grant from the National Key Research and Development Program of China (Project No. 2020YFB1313900), Hong Kong Research Grants Council under General Research Fund (Project No. 14201620), the National Natural Science Foundation of China (Project No. 62006219, 62072452) and Guangdong Provincial Basic and Applied Basic Research Fund-Regional Joint Fund (Project No. 2020B1515130004).

References

[1] Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. Uncertainty-based continual learning with adaptive regularization. *arXiv preprint arXiv:1905.11614*, 2019.

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[3] Sungmin Cha, Hsiang Hsu, Taebaek Hwang, Flavio P Calmon, and Taesup Moon. Cpr: Classifier-projection regularization for continual learning. *arXiv preprint arXiv:2006.07326*, 2020.

[4] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[5] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019.

[6] Pengfei Chen, Guangyong Chen, Junjie Ye, Pheng-Ann Heng, et al. Noise against noise: stochastic label noise helps combat inherent label noise. In *International Conference on Learning Representations*, 2020.

[7] Yu Chen, Tom Diethe, and Neil Lawrence. Facilitating bayesian continual learning by natural gradients and stein gradients. *arXiv preprint arXiv:1904.10644*, 2019.

[8] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. *arXiv preprint arXiv:1906.02425*, 2019.

[9] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR, 2020.

[10] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[11] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[12] Yunhui Guo, Mingrui Liu, Tianbao Yang, and Tajana Rosing. Improved schemes for episodic memory-based lifelong learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[13] Gunshi Gupta, Karmesh Yadav, and Liam Paull. La-maml: Look-ahead meta learning for continual learning. *arXiv preprint arXiv:2007.13904*, 2020.

[14] Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. Overcoming catastrophic forgetting for continual learning via model adaptation. In *International Conference on Learning Representations*, 2018.

[15] Daoyun Ji and Matthew A Wilson. Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature neuroscience*, 10(1):100–107, 2007.

[16] Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International Conference on Machine Learning*, pages 2611–2620. PMLR, 2018.

[17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[19] Quoc V Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *ICML*, 2011.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6391–6401, 2018.

[22] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International Conference on Machine Learning*, pages 3925–3934. PMLR, 2019.

[23] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[24] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*, 2017.

[25] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[26] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. *arXiv preprint arXiv:2006.06958*, 2020.

[27] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5949–5958, 2017.

[28] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[29] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[30] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[31] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[32] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810*, 2018.

[33] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[34] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*, 2021.

[35] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018.

[36] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017.

[37] Stanford. tinyimagenet. `http://cs231n.stanford.edu/tiny-imagenet-200.zip`, 2017.

[38] Binh Tang and David S Matteson. Graph-based continual learning. *arXiv preprint arXiv:2007.04813*, 2020.

[39] Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):1–14, 2020.

[40] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.

[41] Matthew A Wilson and Bruce L McNaughton. Reactivation of hippocampal ensemble memories during sleep. *Science*, 265(5172):676–679, 1994.

[42] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems*, 33, 2020.

[43] Ju Xu and Zhanxing Zhu. Reinforced continual learning. *arXiv preprint arXiv:1805.12369*, 2018.

[44] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. *arXiv preprint arXiv:1902.09432*, 2019.

[45] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[46] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

A Appendix

A.1 PAC Bayesian Bound

In this part, we provide a detailed PAC-Bound based on the continual learning scenario.

Based on the previous works on PAC-Bayes bound [27, 42, 10], let $\ell(\cdot,\cdot)$ be 0-1 loss, then for the empirical loss over the training set $\mathcal{D} \sim \mathscr{D}$, we have $L_{\mathcal{D}}(\boldsymbol{w}) = \frac{1}{|\mathcal{D}|}\sum_{(\boldsymbol{x},y)\in\mathcal{D}}\left[\ell\left(f_{\boldsymbol{w}}\left(\boldsymbol{x}\right),y\right)\right] \in [0,1]$. Given a "prior" distribution P (a common assumption is zero mean, σ^2 variance Gaussian distribution) over the weights of the form $\boldsymbol{w}+\boldsymbol{v}$, the expected error of $f_{\boldsymbol{w}+\boldsymbol{v}}$ can be bounded with probability at least $1-\delta$ over the draw of n training data:

$$\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}}(\boldsymbol{w}+\boldsymbol{v})\right] \leq L_{\mathcal{D}}(\boldsymbol{w}) + \mathbb{E}_{\boldsymbol{v}}\left[L_{\mathcal{D}}(\boldsymbol{w}+\boldsymbol{v})\right] - L_{\mathcal{D}}(\boldsymbol{w}) + 4\sqrt{\frac{1}{n}\left(KL(\boldsymbol{w}+\boldsymbol{v}||P) + \ln\frac{2n}{\delta}\right)}, \tag{6}$$

where $\boldsymbol{w}$ is the weight of the predictor learned from the training set, $\boldsymbol{v}$ is a random variable and is often chosen as a zero mean spherical Gaussian perturbation with variance σ^2 in every direction.

We now consider the bound in the continual learning scenario. To simplify our explanation, we only consider two tasks, which contains the training sets $\mathcal{D}_1$ and $\mathcal{D}_2$ sampled from the distributions $\mathscr{D}_1$ and $\mathscr{D}_2$, respectively. We assume the model $f_{\boldsymbol{w}}$ is learned from the training set $\mathcal{D}_1$ ($\boldsymbol{w} = \operatorname{argmin}_{\boldsymbol{w}} L_{\mathcal{D}_1}(\boldsymbol{w})$, and then continue to learn the training set $\mathcal{D}_2$. Our final goal is to find an optimal parameter $\Delta\boldsymbol{w}$ to minimizes the overall risk $L_{\mathscr{D}_1\cup\mathscr{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w})$ for all tasks as follows:

$$\min_{\Delta\boldsymbol{w}} L_{\mathscr{D}_1\cup\mathscr{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w})$$

Based on Eq. (6), the expected error of $f_{\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}}$ can be bounded with probability at least $1-\delta$ over the draw of training set $\mathcal{D}_2$:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1\cup\mathscr{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] = \mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] + \mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] \\
\leq\ & \underbrace{\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right]}_{\text{stability of old task}} + \underbrace{L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w})}_{\text{sensitivity of new task}} + \underbrace{\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] - L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w})}_{\text{expected sharpness on training set of new task}} \\
&+ 4\sqrt{\frac{1}{n}\left(KL(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}||P) + \ln\frac{2n}{\delta}\right)},
\end{aligned}$$

where n is the size of training set $\mathcal{D}_2$. As we can see, the PAC-Bayes bound in the continual learning scenario depends on four quantities, (1) the stability of old task, (2) the sensitivity of new task, (3) the expected sharpness on training set of new task, and (4) the Kullback Leibler (KL) divergence to the "prior" P. The bound is valid for any distribution measure P, any perturbation distribution $\boldsymbol{v}$ and any method of choosing $\Delta\boldsymbol{w}$ dependent on the training set $\mathcal{D}_2$.

In order to ensure the stability of old task, we constrain $\Delta\boldsymbol{w}$ in the complementary space $\mathcal{V}^C$ of the important space representing the old task $\mathscr{D}_1$ following GPM [34], so that $\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right]$ does not increase too much compared with the previously minimized $\mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1}(\boldsymbol{w}+\boldsymbol{v})\right]$. Let $\boldsymbol{v}\in\mathcal{V}$, we have $\mathbb{E}_{\boldsymbol{v}\in\mathcal{V}}\left[L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] \leq \max_{\boldsymbol{v}\in\mathcal{V}} L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})$, then we can rewrite the above bound as follows:

$$\begin{aligned}
&\min_{\Delta\boldsymbol{w}} \mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1\cup\mathscr{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] \leq \min_{\Delta\boldsymbol{w}\in\mathcal{V}^C} \mathbb{E}_{\boldsymbol{v}}\left[L_{\mathscr{D}_1}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v})\right] + L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}) \\
&+ \max_{\boldsymbol{v}\in\mathcal{V}} L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}) - L_{\mathcal{D}_2}(\boldsymbol{w}+\Delta\boldsymbol{w}) + 4\sqrt{\frac{1}{n}\left(KL(\boldsymbol{w}+\Delta\boldsymbol{w}+\boldsymbol{v}||P) + \ln\frac{2n}{\delta}\right)}.
\end{aligned}$$

Thus, our FS-DGPM exactly optimizes the worst-case of the flatness of the weight loss landscape to control the PAC-Bayes bound, which theoretically justifies both lower loss value and flatter landscape lead to better continual learning performance, and why our proposed FS-DGPM works.

A.2 Derivation for DGPM

We derive the gradient of the importance value λ_i by minimizing the worst performance of $f_{\boldsymbol{w}+\boldsymbol{v}}$ under the current task batch $\hat{\mathcal{D}}_t$ and the batch $\hat{\mathcal{M}}$ sampled from the replay buffer $\mathcal{M}$:

$$\begin{aligned}
\nabla_{\lambda_i} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w}+\boldsymbol{v}) =& \frac{\partial}{\partial\left(\boldsymbol{w}+\boldsymbol{v}\right)} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}+\boldsymbol{v}\right) \cdot \frac{\partial}{\partial \lambda_i}\left(\boldsymbol{w}+\boldsymbol{v}\right) \\
=& \frac{\partial}{\partial\left(\boldsymbol{w}+\boldsymbol{v}\right)} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}+\boldsymbol{v}\right) \cdot \frac{\partial}{\partial \lambda_i}\left(\boldsymbol{w}+\eta_1 \sum_{j=1}^{k} \lambda_j \boldsymbol{u}_j \boldsymbol{u}_j^T \left(\nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t}(\boldsymbol{w})\right)\right) \\
=& \frac{\partial}{\partial\left(\boldsymbol{w}+\boldsymbol{v}\right)} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}+\boldsymbol{v}\right) \cdot \left(\eta_1 \boldsymbol{u}_i \boldsymbol{u}_i^T \left(\nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t}(\boldsymbol{w})\right)\right) \\
\approx& \eta_1 \left(\nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}\right)\right)^T \boldsymbol{u}_i \boldsymbol{u}_i^T \left(\nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t}\left(\boldsymbol{w}\right)\right),
\end{aligned}$$

where $\cdot$ is the dot product operator. Note that we only consider one gradient update to $\boldsymbol{v}$ in the second equation for simplicity, but using multiple gradient updates is a straightforward extension. For the third equation, we get it by assuming that $\boldsymbol{w}$ is constant with respect to λ_i. The last approximation is obtained by the first-order Taylor expansion. Setting all first-order gradient terms as constants to ignore second-order derivatives, we get the approximation as:

$$\nabla_{(\boldsymbol{w}+\boldsymbol{v})} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w}+\boldsymbol{v}) = \nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}\right) + \left(\nabla_{\boldsymbol{w}}^2 L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}\right)\right) \boldsymbol{v} + O\left(\|\boldsymbol{v}\|^2\right) \approx \nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}\left(\boldsymbol{w}\right).$$

The importance of each basis is constrained to be between 0 and 1, where 0 indicates that the basis is not important to old tasks and can completely release for learning new tasks. The initial value of all importance is set to 1, and we use the sigmoid function with a temperature factor of 10 at the end of gradient update: $\lambda_i \leftarrow 1/(1+exp(-10\lambda_i))$.

A.3 Pseudo-code for updating GPM

GPM [34] achieves excellent stability by ensuring that gradient updates only occur in directions orthogonal to the gradient subspaces deemed important for the past tasks. Similar to [34], we calculate the bases of these subspaces for each layer by analyzing network representations after learning each task with Singular Value Decomposition (SVD), and then use it to update $\boldsymbol{v}$ and $\boldsymbol{w}$ by layer.

As shown in Algorithm 2 for updating GPM, we firstly sample n_s random examples from the replay buffer $\mathcal{M}$ to construct the representation matrix for each layer, $\boldsymbol{R}^l$. Next, we perform SVD on $\boldsymbol{R}^l = \boldsymbol{U}^l \Sigma^l (\boldsymbol{V}^l)^T$ and obtained its k-rank approximation $\boldsymbol{R}_k^l$ according to the following **criteria** for the given threshold, ϵ^l:

$$\|\boldsymbol{R}_k^l\|_F^2 \geq \epsilon^l \|\boldsymbol{R}^l\|_F^2, \tag{7}$$

where $\|\cdot\|_F$ is the Frobenius norm of the matrix and and ϵ^l $(0 < \epsilon^l \leq 1)$ is the threshold hyperparameter for layer l.

Algorithm 2 UpdateGPM

Input: Network $f_{\boldsymbol{w}}$ with L-layer, Replay buffer $\mathcal{M}$, sample size n_s, threshold ϵ for each layer.
Result: Bases matrix $\{(\boldsymbol{M}_l)_{l=1}^L\}$ ▷ till $L-1$ if multi-head setting.
if $\mathcal{M}$ is not empty **then**
  $B_{n_s} \sim \mathcal{M}$ ▷ sample a mini-batch of size n_s from $\mathcal{M}$.
  $\mathcal{R} \leftarrow$ forward$(B_{n_s}, f_{\boldsymbol{w}})$, where $\mathcal{R} = \{(\boldsymbol{R}^l)_{l=1}^L\}$ ▷ construct representation matrix for each layer by forward pass.
  for $l = 1, 2, \cdots, L$ **do**
    $\boldsymbol{U}^l \leftarrow \mathrm{SVD}(\boldsymbol{R}^l)$ ▷ update new bases for each layer by performing SVD.
    $k \leftarrow$ criteria $(\boldsymbol{R}^l, \epsilon^l)$ ▷ see Eq. (7).
    $\boldsymbol{M}_l \leftarrow \boldsymbol{U}^l[0:k]$
  end for
end if

For fully connected layer, the representation matrix $\boldsymbol{R}^l = [\boldsymbol{x}_1^l, \boldsymbol{x}_2^l, \cdots, \boldsymbol{x}_{n_s}^l]$ concatenates n_s inputs of the l-th layer linear function along the column, which are obtained by forwarding the batch of n_s samples $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{n_s}\}$ through the network $f_{\boldsymbol{w}}$. For convolution (Conv) layers, we first express the Conv as matrix multiplication by reshaping the input tensor $\mathcal{X} \in \mathbb{R}^{C_i \times h_i \times w_i}$ and filters $\mathcal{W} \in \mathbb{R}^{C_o \times C_i \times k \times k}$ into $\boldsymbol{X} \in \mathbb{R}^{(h_o \times w_o) \times (C_i \times k \times k)}$ and $\boldsymbol{W} \in \mathbb{R}^{(C_i \times k \times k) \times C_o}$ respectively, where $C_i(C_o)$ denotes the number of input (output) channels of the Conv layer, $h_i, w_i(h_o, w_o)$ represents the height and width of the input (output) feature maps and k is the kernel size of the filters. Then we construct the representation matrix as $\boldsymbol{R}^l = [(\boldsymbol{X}_1^l)^T, (\boldsymbol{X}_2^l)^T, \cdots, (\boldsymbol{X}_{n_s}^l)^T] \in \mathbb{R}^{(C_i \times k \times k) \times (h_o \times w_o \times n_s)}$. The key difference with GPM [34] is that we perform SVD in the entire gradient space and use the obtained bases to replace the last calculated bases, while [34] obtains the newly added bases by performing SVD in the subspace orthogonal to the existing bases. In addition, GPM can be regarded as a special case of our method when η_1 and η_2 are set to 0.

B Details for Landscape Visualization for Continual Learning

In this section, we first provide the pseudo-code of the visualization of the weight loss landscape in continual learning, and then provide more empirical results.

B.1 Pseudo-code for Visualization

As shown in Algorithm 3 for the visualization of the weight loss landscape for the continual learning scenario, we first sample a random direction $\boldsymbol{d}$ from a Gaussian distribution, and then apply the filter-wise normalization following [21] to eliminate the scaling invariance of DNNs. Next, we independently calculate the training loss of a series of perturbed weights for each learned task. For a given task descriptor τ and perturbed weights $\boldsymbol{w} + \alpha\boldsymbol{d}$, we obtain the training loss of the perturbed model $f_{\boldsymbol{w}+\alpha\boldsymbol{d}}$ on all training samples of task τ. Then, we plot the weight loss landscape for task τ. If the descriptor for the current training task of the model is t, we will plot t curves.

Algorithm 3 Visualization of the Weight Loss Landscape

Input: Network $f_{\boldsymbol{w}}$ with L-layer (F_l filters in the l-th layer), current task descriptor t, training dataset $\mathcal{D}_\tau = \{\boldsymbol{x}_{i,\tau}, y_{i,\tau}\}_{i=1}^{n_\tau}$ for $\tau = 1, ..., t$, the scalar parameter $\alpha \in [\alpha_{min}, \alpha_{max}]$.
Sample a random direction $\boldsymbol{d} \sim \mathcal{N}(0, 1)$
for $l = 1, 2, \cdots, L$ **do**
  for $j = 1, 2, \cdots, F_l$ **do**
    $\boldsymbol{d}_{l,j} \leftarrow \frac{\boldsymbol{d}_{l,j}}{\|\boldsymbol{d}_{l,j}\|_F} \|\boldsymbol{w}_{l,j}\|_F$ ▷ filter-wise normalization for the random direction.
  end for
end for
for $\tau = 1, 2, \cdots, t$ **do**
  for $\alpha = \alpha_{min}, \cdots, \alpha_{max}$ **do**
    $L_{\mathcal{D}_\tau}(\boldsymbol{w} + \alpha\boldsymbol{d}) = \frac{1}{n_\tau} \sum_{(\boldsymbol{x},y) \in \mathcal{D}_\tau} \ell\left(f_{\boldsymbol{w}+\alpha\boldsymbol{d}}(\boldsymbol{x}), y\right)$ ▷ calculate training loss of the perturbed model on task τ.
  end for
  Plot$(\alpha, L_{\mathcal{D}_\tau}(\boldsymbol{w} + \alpha\boldsymbol{d}))$, $\forall \alpha \in [\alpha_{min}, \alpha_{max}]$
end for

B.2 More Results for Section 3.2: Connection of Weight Loss Landscape and Continual Learning

To investigate the relationship between the weight loss landscape and stability-sensitivity in the continual learning scenario, we use the previously proposed GPM [34], La-MAML [13], and ER [5] to train an MLP network with two hidden layers on the Permuted MNIST (PMNIST) [20] dataset that contains 10 tasks. For each task, we use $60,000$ training samples instead of $1,000$ used in the experimental environment. The replay buffer size is set to $1,000$. We also use Oracle and Finetune, which respectively represent retraining the network on the entire dataset contain all passed tasks, and training the network on the data stream without any regularization or episodic memory. Considering the direction $\boldsymbol{d}$ for visualization is randomly selected, we repeat the visualization 10 times with different $\boldsymbol{d}$. Figure 10, 11, 12, 13, and 14 show the weight loss landscape for each task when a new

task is trained using Oracle, Finetune, GPM, La-MAML, and ER, respectively. The i-th row of each figure represents changes in the weight loss landscape of the i-th task during the model evolution, and each j-th column indicates that the current model has learned j tasks.

C Details for FS-ER

In this section, we first provide the pseudo-code of Flattening Sharpness for Vanilla ER, and then provide more empirical results.

C.1 Pseudo-code for FS-ER

Comparing the pseudo-code of Vanilla ER, FS-ER only adds the adversarial weight perturbation $\boldsymbol{v}$.

Algorithm 4 FS-ER

Input: Network weight $\boldsymbol{w}$, loss function ℓ, learning rate η_3, FS step size η_1, FS steps K, batch size b.
Initializing $\mathcal{M} \leftarrow \{\}$
for $t = 1, 2, \cdots, T$ **do**
  for $ep = 1, 2, \cdots, num_{epochs}$ **do**
    for batch $\hat{\mathcal{D}}_t \overset{b}{\sim} \mathcal{D}_t$ **do** ▷ Sample without replacement a mini-batch from task t.
      $\hat{\mathcal{M}} \overset{b}{\sim} \mathcal{M}$ ▷ Sample a mini-batch from $\mathcal{M}$.
      for $k = 1, \cdots, K$ **do**
        $\boldsymbol{v} \leftarrow \boldsymbol{v} + \eta_1 \nabla_{(\boldsymbol{w}+\boldsymbol{v})} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w} + \boldsymbol{v})$ ▷ Add adversarial weight perturbation $\boldsymbol{v}$.
      end for
      $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_3 \nabla_{\boldsymbol{w}} L_{\hat{\mathcal{D}}_t \cup \hat{\mathcal{M}}}(\boldsymbol{w} + \boldsymbol{v})$
      Push $\hat{\mathcal{D}}_t$ to $\mathcal{M}$ with reservior sampling
    end for
  end for
end for

C.2 More Results for Section 3.3: A Case Study of Flattening Sharpness for Vanilla ER

Figure 14 and 15 show the weight loss landscape for each task when a new task is trained using ER and FS-ER, respectively.

D Experimental Details

D.1 Datasets

Table 4 summarizes the statistics of four datasets used in our experiments.

Table 4: Dataset Statistics.

	PMNIST	**CIFAR-100 Split**	**CIFAR-100 Superclass**	**TinyImageNet**
Input size	1×28×28	3×32×32	3×32×32	3×64×64
# tasks	20	10	20	40
# Classes/task	10	10	5	5
# Training/task	1,000	4,750	2,375	2,250
# Validation/task	-	250	125	250
# Test/task	10,000	1,000	500	250

D.2 Architecture

AlexNet-like architecture: For 10-Split CIFAR-100, similar to GPM [34], we use a AlexNet-like architecture with three convolutional (Conv) layers and two fully connected layers. The three Conv layers have 64, 128, and 256 filters with 4×4, 3×3, and 2×2 kernel sizes, respectively. Both fully connected layers have 2048 units in each layer. Max-pooling layer with filters of size 2×2 is used after the Conv layer. Dropout of 0.2 is used for the first two layers, and 0.5 is used for the remaining layers. Batch normalization is only used in the second layer.

Modified Lenet-5 architecture: For 20-Spilt CIFAR-100 Superclass, similar to GPM [34], we use a modified LeNet-5 architecture with 20-50-800-500 neurons, of which the first two are Conv layers with 5×5 kernel sizes, and the last two are fully connected layers. Batch normalization and max-pooling layer with filters of size 3×3 with a stride of 2 are used in the Conv layers. Batch normalization parameters are learned for the first task and shared with all the other tasks.

Architecture for TinyImageNet: For 40-Spilt TinyImageNet, similar to La-MAML [13], we use a CNN having 4 Conv layers with 160 3×3 filters. The output from the final Conv layer is flattened and is passed through 2 fully connected layers having 320 and 640 units, respectively.

All networks use ReLU in the hidden units, and finally have a multi-head output layer to perform classification for every task. No bias units are used following [34].

D.3 Baselines

We compare our method against multiple methods described below.

- **EWC** [17]: Elastic Weight Consolidation is a regularization-based method that uses the diagonal of Fisher information to identify important weights.
- **ICARL** [30]: ICARL is a memory-based method that uses knowledge distillation and episodic memory to reduce forgetting.
- **GEM** [24]: Gradient Episodic Memory uses the gradient of episodic memory to constrain optimization to make sure that the gradients of the new task do not change the previous knowledge.
- **ER** [31, 5]: Experience Replay uses a small replay buffer to store old data using reservoir sampling. Then, the stored data is replayed again with the new data samples.
- **La-MAML** [13]: Look-ahead MAML is inspired by optimization-based meta-learning that leverages replay to avoid forgetting and favor positive backward transfer by asynchronously learning the weights and LRs.
- **GPM** [34]: Gradient Projection Memory minimizes forgetting by taking gradient steps orthogonal to the gradient subspace deemed important for the past tasks when learning a new task.
- **Multitask**: Multitask is an oracle baseline that all tasks are learned jointly using the entire dataset at once in a single network. Multitask is not a continual learning strategy but serves as an upper bound on average test accuracy on all tasks.

GEM [24], ICARL [30], and La-MAML [13] are implemented from the official implementation provided by [13]. EWC [17] is implemented from the official implementation provided by [24]. GPM [34] is implemented from its official implementation. ER [31, 5] is implemented by adapting the code provided by [31].

D.4 GPU Device

We measured the average training time per task calculated on the NVIDIA GeForce RTX 2080 Ti GPU. Figure 5(b) shows the training time per task on the 20-Split CIFAR-100 Superclass experiment using the baselines and our method.

D.5 Threshold Hyperparameter

Following [34], we use a different value of the threshold hyperparameter, ϵ, for different architectures and different datasets. For PMNIST experiment, we use ϵ = 0.99 for all the layers and increasing the value of ϵ by 0.0005 for each new task. For CIFAR-100 Split and CIFAR-100 Superclass experiment, we use the values reported by [34]. For CIFAR-100 Split, the initial value of ϵ is 0.97 for all the layers and increasing by 0.003 for each new task. For CIFAR-100 Superclass experiment, we use ϵ = 0.98 for all the layers and increasing by 0.001 for each new task. For TinyImageNet experiment, we use ϵ = 0.9 for all the layers and increasing by 0.0025 for each new task.

D.6 List of Hyperparameters

We report in Table 5 the hyper-parameters selected by grid-search for all baselines and our method. For PMNIST, we use a hyper-parameter called glances for all compared approaches and set it to 5 following [13]. This hyper-parameters indicates the number of meta-updates made on each incoming sample of data. In addition, for each task in PMNIST, we use 5 epochs to train the network for Multitask instead of 1 epoch in other methods.

Table 5: List of hyperparameters for the baselines and our approach. "LR" denotes the (initial) learning rate. Superclass is the abbr. of CIFAR-100 Superclass.

Method	Parameter	**PMNIST**	**CIFAR-100 Split**	**Superclass**	**TinyImageNet**
EWC	LR	0.01	0.005	0.03	-
	memory strength, γ	100	5000	1000	-
ICARL	LR	-	0.03	0.01	-
	memory strength, γ	-	0.1	0.5	-
	memory size	-	1000	1000	-
GEM	LR	0.01	0.01	0.03	-
	memory strength, γ	0.0	0.5	0.5	-
	memory size	200	1000	1000	-
ER	LR	0.005	0.03	0.01	-
	memory size	200	1000	1000	-
La-MAML	LRs, α_0	0.15	0.1	0.1	-
	LR for LRs, η	0.3	0.5	0.5	-
	memory size	200	1000	1000	-
GPM	LR	0.01	0.01	0.01	0.005
	n_s	300	125	125	200
Multitask	LR	0.01	0.01	-	-
FS-DGPM	LR, η_3	0.01	0.01	0.01	0.01
	LR for sharpness, η_1	0.05	0.001	0.01	0.001
	LR for DGPM, η_2	0.01	0.01	0.01	0.01
	memory size	200	1000	1000	400
	n_s	200	125	125	200

E Additional Experimental Results

We provide the results of different step numbers K in solving weight perturbation $\boldsymbol{v}$. We evaluation FS-DGPM with $K \in \{1, 2, 3\}$ in the CIFAR-100 Split experiment. As shown in Figure 7, two steps have been well improved, and the extra steps only bring few improvements but with much more time.

We also compare ER and our method on CIFAR-100 Superclass when the memory size is 100, 500, and 1000. As shown in Figure 8, we see that both ER and FS-DGPM benefits from increases in memory size, but the outperformance of FS-DGPM is more visible under the low-resource regime. We think the advantageous performance of FS-DGPM can be attributed to the effective utilization of episodic memory converted into bases through SVD.

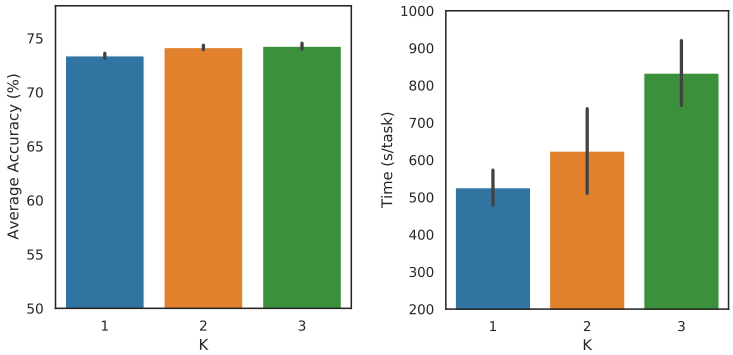

Figure 7: Results of different step numbers K for weight perturbation $\boldsymbol{v}$ on CIFAR-100 Split in 50 epochs. Each experiment is run with 5 seeds.

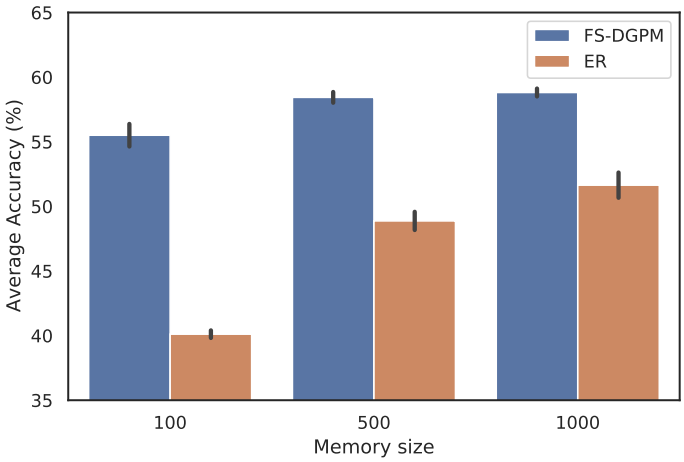

Figure 8: Average test accuracy of different memory size on CIFAR-100 SuperClass. Each experiment is run with 5 seeds.

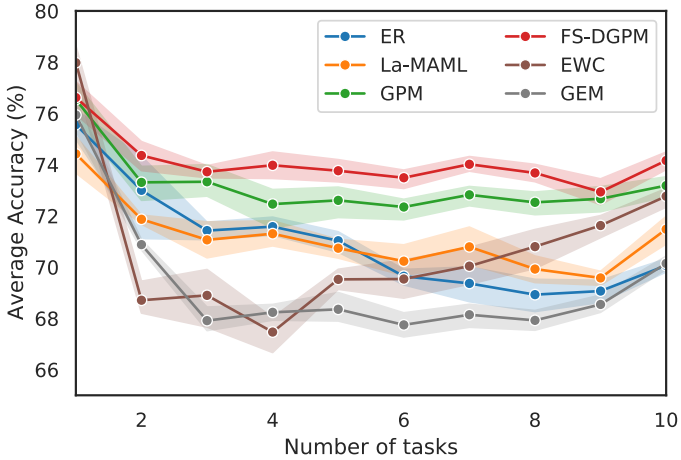

Figure 9: Average test accuracy as a function of the number of tasks trained on 10-Split CIFAR-100. Each experiment is run with 5 seeds.

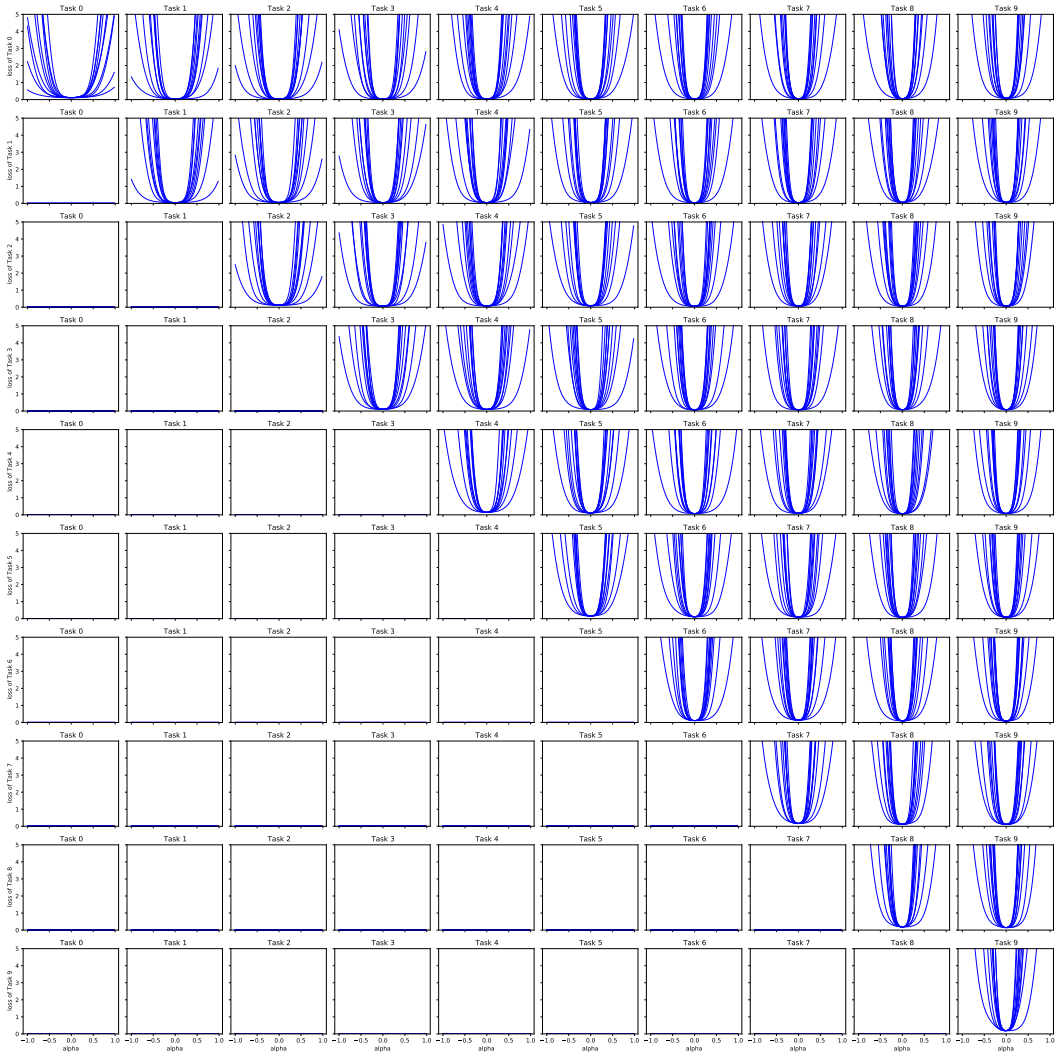

Figure 10: The weight loss landscape of the ten tasks of PMNIST using Oracle for training. The i-th row represents changes in the weight loss landscape of the i-th task during the model evolution, and the j-th column indicates that the model has learned j tasks. The y-axis is the loss value, and the x-axis is the scalar value for visualization random direction. (Task t is the abbr. of $t+1$-th task.)

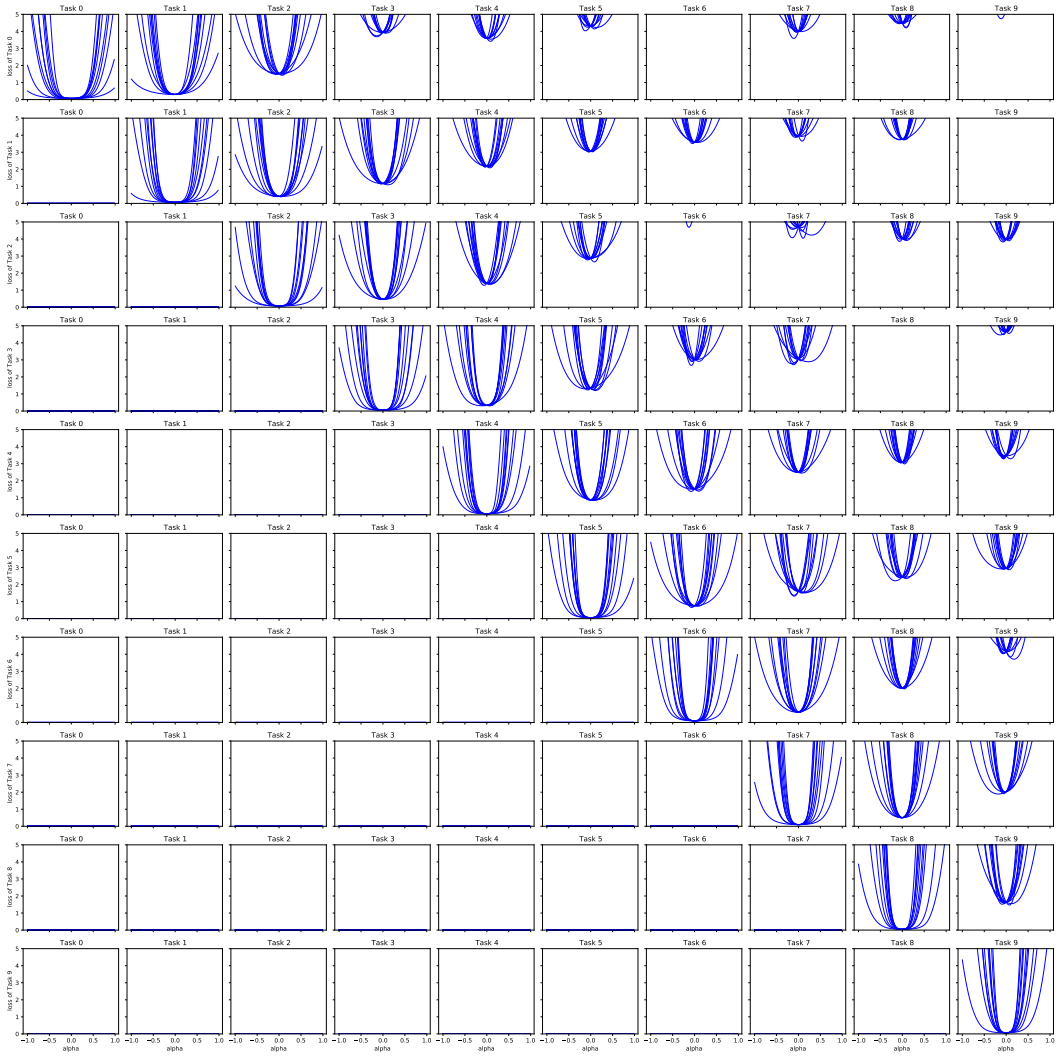

Figure 11: The weight loss landscape of the ten tasks of PMNIST using Finetune for training. The i-th row represents changes in the weight loss landscape of the i-th task during the model evolution, and the j-th column indicates that the model has learned j tasks. The y-axis is the loss value, and the x-axis is the scalar value for visualization random direction. (Task t is the abbr. of $t+1$-th task.)

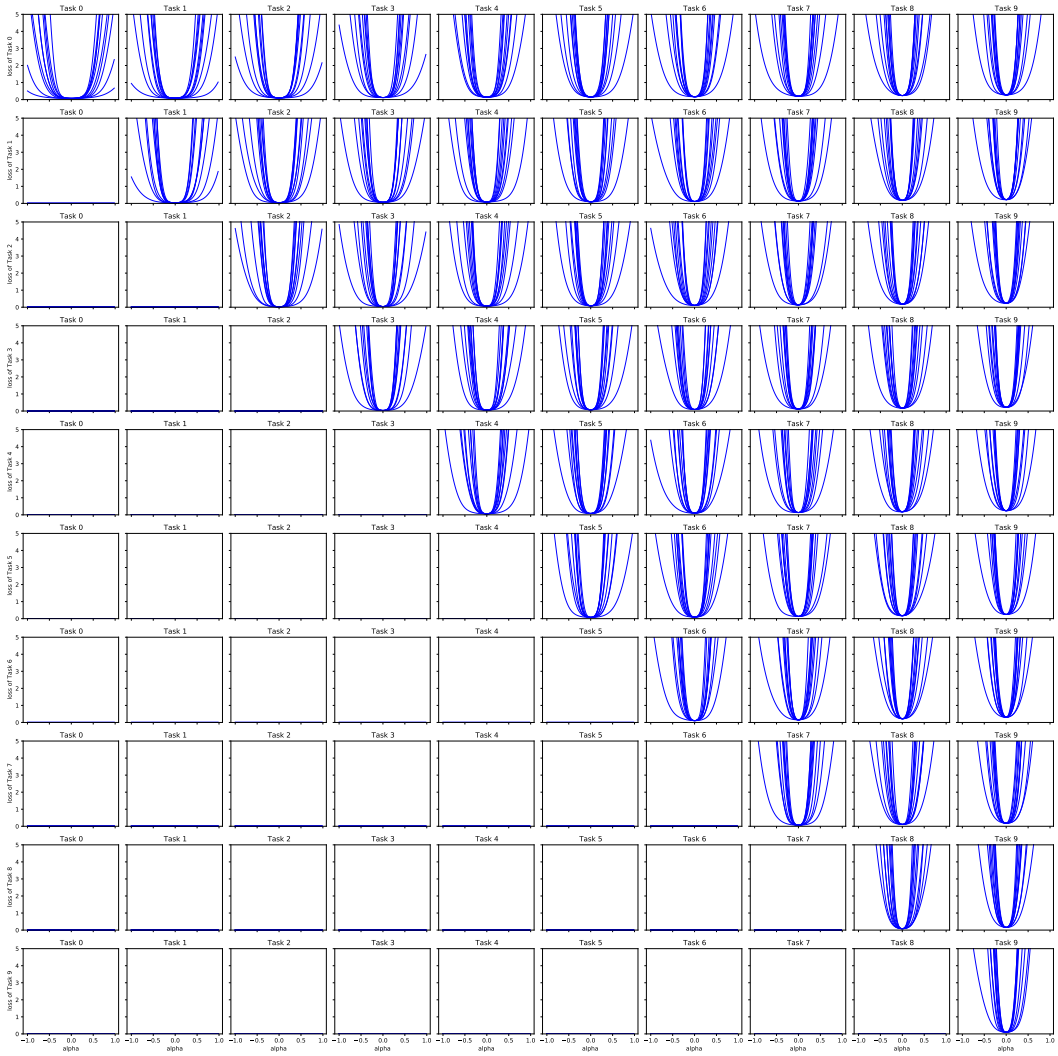

Figure 12: The weight loss landscape of the ten tasks of PMNIST using GPM for training. The i-th row represents changes in the weight loss landscape of the i-th task during the model evolution, and the j-th column indicates that the model has learned j tasks. The y-axis is the loss value, and the x-axis is the scalar value for visualization random direction. (Task t is the abbr. of $t+1$-th task.)

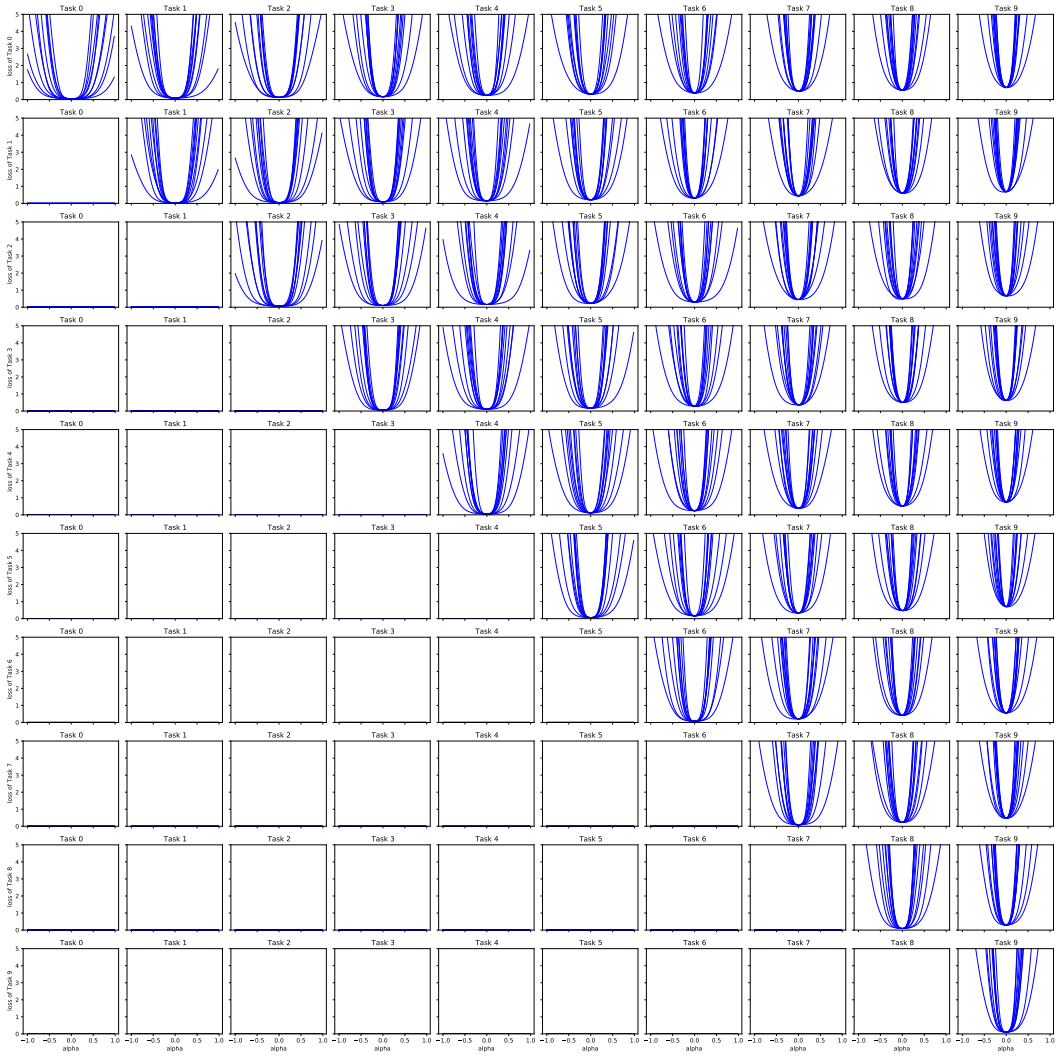

Figure 13: The weight loss landscape of the ten tasks of PMNIST using La-MAML for training. The i-th row represents changes in the weight loss landscape of the i-th task during the model evolution, and the j-th column indicates that the model has learned j tasks. The y-axis is the loss value, and the x-axis is the scalar value for visualization random direction. (Task t is the abbr. of $t+1$-th task.)

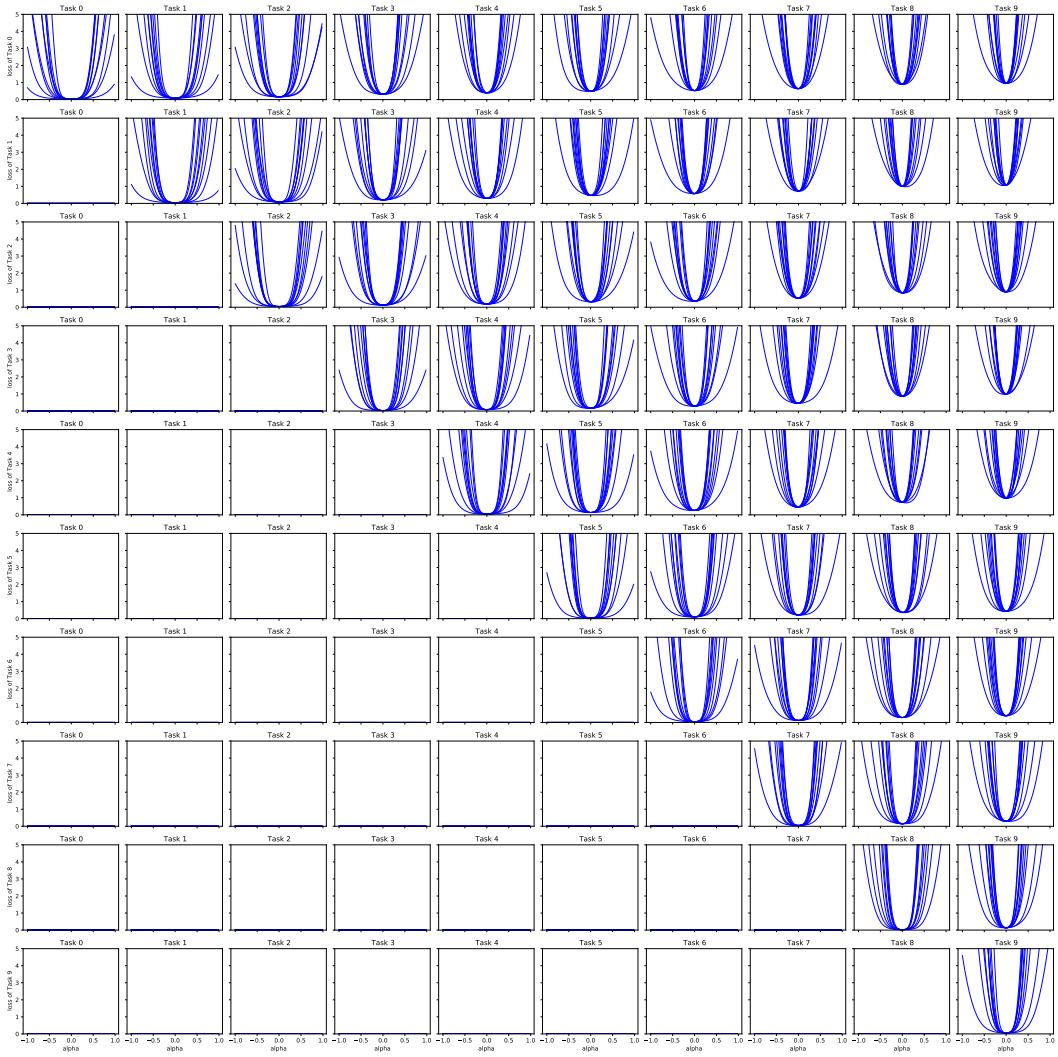

Figure 14: The weight loss landscape of the ten tasks of PMNIST using ER for training. The i-th row represents changes in the weight loss landscape of the i-th task during the model evolution, and the j-th column indicates that the model has learned j tasks. The y-axis is the loss value, and the x-axis is the scalar value for visualization random direction. (Task t is the abbr. of $t+1$-th task.)

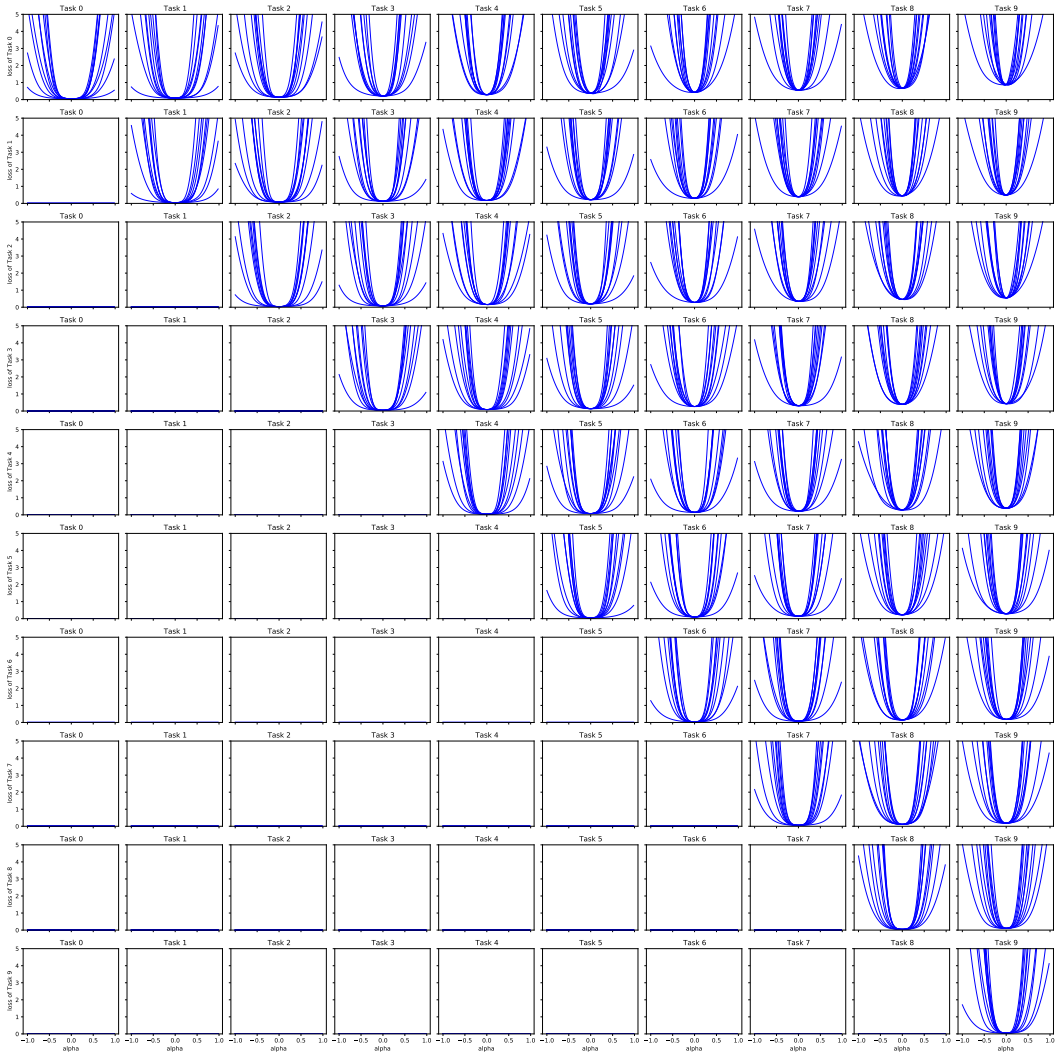

Figure 15: The weight loss landscape of the ten tasks of PMNIST using FS-ER for training. The i-th row represents changes in the weight loss landscape of the i-th task during the model evolution, and the j-th column indicates that the model has learned j tasks. The y-axis is the loss value, and the x-axis is the scalar value for visualization random direction. (Task t is the abbr. of $t+1$-th task.)

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Refer to Section 6 for details.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the GitHub link of our code in the camera-ready version.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix D for more details.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D.4 for more details.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide the GitHub link of our code in the camera-ready version.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]