Data Augmented Flatness-aware Gradient Projection for Continual Learning

Enneng Yang[1] Li Shen[2*] Zhenyi Wang[3*] Shiwei Liu[4] Guibing Guo[1*] Xingwei Wang[1]
[1]Northeastern University, China [2]JD Explore Academy, China
[3]University of Maryland, USA [4]The University of Texas at Austin, USA
ennengyang@stumail.neu.edu.cn, {mathshenli,wangzhenyineu}@gmail.com,
shiwei.liu@austin.utexas.edu, {guogb,wangxw}@swc.neu.edu.cn

Abstract

*The goal of continual learning (CL) is to continuously learn new tasks without forgetting previously learned old tasks. To alleviate catastrophic forgetting, gradient projection based CL methods require that the gradient updates of new tasks are orthogonal to the subspace spanned by old tasks. This limits the learning process and leads to poor performance on the new task due to the projection constraint being too strong. In this paper, we first revisit the gradient projection method from the perspective of flatness of loss surface, and find that unflatness of the loss surface leads to catastrophic forgetting of the old tasks when the projection constraint is reduced to improve the performance of new tasks. Based on our findings, we propose a **D**ata Augmented **F**latness-aware **G**radient **P**rojection (DFGP) method to solve the problem, which consists of three modules: data and weight perturbation, flatness-aware optimization, and gradient projection. Specifically, we first perform a flatness-aware perturbation on the task data and current weights to find the case that makes the task loss worst. Next, flatness-aware optimization optimizes both the loss and the flatness of the loss surface on raw and worst-case perturbed data to obtain a flatness-aware gradient. Finally, gradient projection updates the network with the flatness-aware gradient along directions orthogonal to the subspace of the old tasks. Extensive experiments on four datasets show that our method improves the flatness of loss surface and the performance of new tasks, and achieves state-of-the-art (SOTA) performance in the average accuracy of all tasks.*

1. Introduction

Humans can learn a series of continuously encountered tasks without forgetting previously learned knowledge. In recent years, many researches target for making the neural network achieve continual learning (CL) ability like humans [38, 47]. One of the main challenge of CL is to mitigate the catastrophic forgetting [40, 17, 53] of the knowledge of previous tasks when learning new tasks [51].

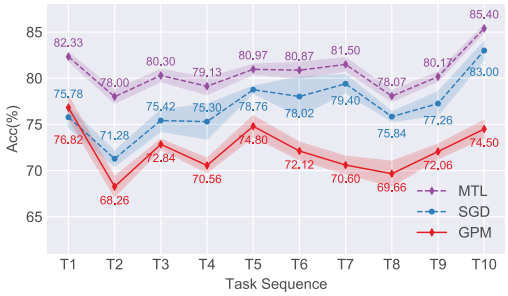

Figure 1: New tasks' accuracy (Higher Better) of GPM, SGD, and MTL on CIFAR-100 dataset, where a significant performance gap exists between GPM and SGD/MTL.

To alleviate catastrophic forgetting of old tasks, several works [57, 15, 8, 27, 43] propose to constrain the gradient update direction of the new tasks. The new tasks only update the network along the orthogonal direction to the gradient subspaces deemed for the old tasks. Compared to other methods, the recently proposed Gradient Projection Memory (GPM) [43] shows better performance in CL. However, we observe poor performance for the new tasks in GPM. As shown in Fig. 1, by comparing vanilla SGD (which learns new tasks without any gradient constraints) or MTL (which learns all tasks simultaneously and can be seen as an upper bound for CL) with GPM, we find that the performance of new tasks with GPM has a large gap compared with SGD and MTL. When learning the 10-th task $T10$, MTL and SGD can achieve 85.40% and 83.00% accuracy, respectively, while GPM can only achieve 74.50%. This vast gap has prompted us to explore the reasons behind it and devise effective ways to address this issue.

In this paper, we first revisit the 'stability-plasticity' dilemma [33] in GPM from the perspective of the flatness of loss surface (we provide the formal definition of flatness in the appendix). We find that the projection threshold in GPM is a key factor in improving the performance of new

*Corresponding author.

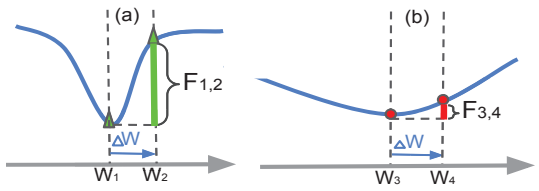

Figure 2: An illustration of (a) a sharp loss surface and (b) a flat loss surface. The blue curve represents the loss surface of task T, and the length of the blue arrow ($\Delta \boldsymbol{W}$) represents the amount of parameter updates for the new task $T+1$.

tasks, as shown in Fig. 3. Specifically, reducing the projection threshold can improve the performance of new tasks. However, it leads to catastrophic forgetting of old tasks and a drop in average performance across all tasks. We suspect that catastrophic forgetting is because the network's loss surface is not flat enough, which is defined as regions on loss surfaces where the loss changes slowly with the model parameters [20]. Fig. 2 gives an example, under the same amount of update $\Delta \boldsymbol{W}$, when the minimum $\boldsymbol{W}_3$ of a flat loss surface and the minimum $\boldsymbol{W}_1$ of a sharp loss surface is moved to $\boldsymbol{W}_4$ and $\boldsymbol{W}_2$, respectively, the loss change $F_{3,4}$ of the former is much smaller than the loss change $F_{1,2}$ of the latter. In other words, a flat loss surface effectively reduces the degree of catastrophic forgetting. We verify in Fig. 4 that catastrophic forgetting of old tasks at a low projection threshold in GPM is indeed due to unflatness loss surfaces. This inspires us to realize that when the loss surface is flat, we can reduce the projection threshold in GPM to improve the performance of new tasks, while simultaneously ensuring that old tasks are not catastrophically forgotten.

To achieve this goal, we propose a novel **D**ata Augmented **F**latness-aware **G**radient **P**rojection (DFGP) method whose critical insight is to encourage a flat loss surface for the CL model. DFGP consists of three modules: data and weight perturbation, flatness-aware optimization, and gradient projection. Specifically, DFGP first performs a flatness-aware mixup with raw training data interpolation to cover potential unseen regions. Data augmentation could be considered a worst-case perturbation w.r.t. training data, which makes the model more robust against distribution shifts (which is the greatest characteristic of CL tasks) and provides a wider exploration (data) space for the subsequent flatness-aware optimization. At the same time, we perturb the current weight in a fixed-radius neighborhood to explore where the loss function is worst-case w.r.t. the weight. Then, under the original training data and the worst-case perturbed data, the loss and the flatness of loss surface are optimized to obtain a flatness-aware gradient. Finally, DFGP uses the orthogonal projection to update the network parameters. Since the loss surface of old tasks in DFGP is flat, DFGP allows new tasks to update the network under relatively smaller projection threshold constraints while keeping the forgetting degree of old tasks close to GPM, greatly improving the performance of new tasks and the overall performance of all tasks.

Empirically, on the four widely used CL datasets, PMNIST, CIFAR-100, 5-Datasets and MiniImageNet, DFGP improves the accuracy by 1.46%, 2.28%, 0.97%, and 8.93%, respectively, compared with GPM. This is significant for these benchmark datasets. It is worth mentioning that a byproduct of our DFGP is that it is more robust to adversarial examples than GPM, since the perturbation of training data in DFGP increases the chance of covering noisy samples. In addition, our proposed data-augmented flatness-aware optimization can be effortlessly combined with other CL methods (such as regularization-based and memory-based methods) to further boost their performance.

The main contributions can be summarized as four-fold:

- We revisit the GPM approach from the perspective of loss surface flatness, and find that when we improve the performance of new tasks, catastrophic forgetting of old tasks occurs due to the loss surface unflatness.
- We propose a data augmented flatness-aware gradient projection (DFGP) method for CL, which simultaneously optimizes the loss and the flatness of the loss surface from both the data and weight levels.
- We compare the proposed DFGP with multiple CL methods on four widely used benchmark datasets and verify that DFGP improves the performance of new tasks and the average performance of all tasks.
- We demonstrate that DFGP is more robust to adversarial attacks relative to GPM, and our optimization strategy can be effortlessly combined with other CL methods to further improve their performance.

2. Rethinking the GPM Method

In this section, we first introduce the problem setup of CL and the Gradient Projection Memory (GPM) method, and then revisit the poor performance of the GPM method on the new arriving task, i.e., the plasticity of GPM.

2.1. Problem Setup

CL [38, 47] is trained with sequential arriving tasks $\{\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(T)}\}$, where $\mathcal{D}^{(t)} = \{(\boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)})\}$ is the training data of task t. When learning the t-th task, we only have access to the data $\mathcal{D}^{(t)}$ for the t-th task. CL expects that a neural network with L layers $f(\boldsymbol{W}, \cdot)$ to be capable of learning these sequentially encountered tasks and not forgetting previously learned tasks. The weight of the network is denoted as $\boldsymbol{W} = \{(\boldsymbol{W}^l)_{l=1}^L\}$ and $\boldsymbol{W}^l$ is the weight of l-th layer. Given an input data $x_i^{(t)} \in \boldsymbol{X}^{(t)}$, denote the input and output of the l-th layer of the network as $x_i^{(t),l}$ and $x_i^{(t),l+1} = f(\boldsymbol{W}^l, x_i^{(t),l})$, respectively, where $x_i^{(t),l}$ is denoted as the **representations** of $x_i^{(t)}$ of task t at layer l. CL expects to minimize the loss of all tasks. According to the Empirical Risk Minimization (ERM) principle [49], we can obtain the

objective of CL as follows:

$$\min_{\boldsymbol{W}} \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{D}^{(t)}|}\sum_{(x_i^{(t)},y_i^{(t)})\in\mathcal{D}^{(t)}}\mathcal{L}^t\left(f(\boldsymbol{W},x_i^{(t)}),y_i^{(t)}\right),$$

where $\mathcal{L}^t(\cdot)$ represents the loss function of the t-th task, e.g., mean squared or cross-entropy loss.

2.2. GPM Method

Since CL can only use the data of task t when learning t-th task, the optimization objective is: $\min_{\boldsymbol{W}} \frac{1}{|\mathcal{D}^{(t)}|}\sum_{(x_i^{(t)},y_i^{(t)})\in\mathcal{D}^{(t)}}\mathcal{L}^t(f(\boldsymbol{W},x_i^{(t)}),y_i^{(t)})$. To avoid catastrophic forgetting of old tasks, the weight $\{(\boldsymbol{W}^l)_{l=1}^L\}$ on task t is updated along the direction orthogonal to the subspace S^l spanned by the previous $t-1$ tasks in gradient projection based CL methods [15, 57, 27, 43]. In particular, GPM [43] has achieved remarkable results compared to other approaches. For task t, the update rule of l-th layer ($l\in\{1,2,\ldots,L\}$) weight $\boldsymbol{W}^l$ in GPM is as follows:

$$\boldsymbol{W}^l = \boldsymbol{W}^l - \eta\cdot(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t)} - \text{Proj}_{S^l}(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t)})), \quad (1)$$

where $\boldsymbol{g}_{\boldsymbol{W}^l}^{(t)} = \nabla\mathcal{L}^t(f(\boldsymbol{W}^l,x_i^{(t)}),y_i^{(t)})$ represents the gradient of the training sample $(x_i^{(t)},y_i^{(t)})$ with respect to the weight $\boldsymbol{W}^l$, η is the step size, and $\text{Proj}_{S^l}(\cdot)$ is a projection operator that projects $\boldsymbol{g}_{\boldsymbol{W}^l}^{(t)}$ into Core Gradient Space (CGS), i.e., subspace S^l spanned by old tasks. The core operation of GPM is to find and store the basis of the old task's CGS. Specifically, when task t is learned, it first randomly samples n samples to obtain the representation matrix $\boldsymbol{R}^{(t),l}=[x_1^{(t),l},\ldots,x_n^{(t),l}]$ of l-th layer of the CL network. Then, it performs SVD [11] on $\boldsymbol{R}^{(t),l}=\boldsymbol{U}^{(t),l}\boldsymbol{\Sigma}^{(t),l}\boldsymbol{V}^{(t),l}$ and approximates its k-rank $\boldsymbol{R}_k^{(t),l}$ according to the following projection criteria for a given threshold ϵ_{th}^l (we call it **projection threshold**): $\|\boldsymbol{R}_k^{(t),l}\|_F^2 \geq \epsilon_{th}^l\|\boldsymbol{R}^{(t),l}\|_F^2$. Next, GPM defines the feature vectors corresponding to the top-k maximum singular values as the basis of the l-th layer, and constitutes bases $\boldsymbol{M}^l$ of the space S^l by them: $\boldsymbol{M}^l = span\{u_1^{(t),l},\ldots,u_k^{(t),l}\}$. Finally, the calculation rule of the projection operation is as follows: $\text{Proj}_{S^l}(\boldsymbol{g}^{(t),l}) = \boldsymbol{M}^l(\boldsymbol{M}^l)^\top\boldsymbol{g}^{(t),l}$.

2.3. Rethinking the Plasticity in GPM

Regretfully, GPM still performs poorly on new tasks (see Fig. 1), limiting its overall performance. Below, we try to explore what limits the performance of GPM.

Which factor influences the performance of new tasks in GPM? There exist distribution drifts between the tasks encountered continuously in CL. When the constraints imposed by new tasks in CL are too strong, it causes the network to fail to fit the distribution of the new tasks, resulting in poor performance. In particular, the factor that determines the strength of constraints in GPM is the *projection threshold* ϵ_{th}^l. We show in Fig. 3 (red line) that a relatively lower projection threshold can achieve better performance on new tasks. Specifically, when we reduce the projection threshold

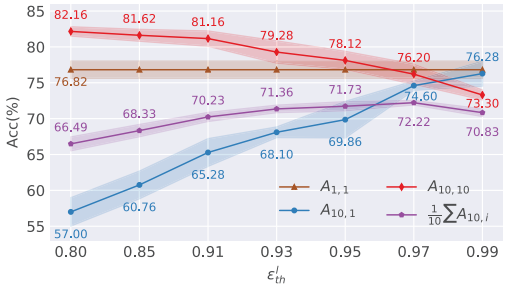

Figure 3: The accuracy of an old task ($T1$, blue line), a new task ($T10$, red line) and the average accuracy of all tasks (purple line) under different projection threshold ϵ_{th}^l of GPM method on CIFAR-100. $A_{t,i}$ represents the accuracy of the model tested on task i after the trained task t.

from 0.99 to 0.80, the performance $A_{10,10}$ of the new task $T10$ improves from 73.30% to 82.16%.

What are the problems with increasing the performance of new tasks in GPM? Although a relatively small threshold leads to better performance on new tasks, it also leads to catastrophic forgetting of old tasks, resulting in overall performance degradation. Specifically, as shown in Fig. 3 (brown line), task $T1$ can achieve 76.82% accuracy when it is learned. However, after learning task $T10$, the accuracy $A_{10,1}$ (blue line) of task $T1$ drops to 76.28%, 57.00% at two projection thresholds of 0.99 to 0.80.

Is it possible to improve the performance of new tasks without catastrophic forgetting of old tasks? According to previous analysis for GPM, when we want to achieve high performance for new tasks by setting a relatively small projection threshold, we need to ensure that old tasks are not catastrophically forgotten. A flat loss surface [20, 16, 22] seems to offer a chance that old tasks are not catastrophically forgotten. An example is given in Fig. 2, after task T is learned, the weight is $\boldsymbol{W}_1$ (a minimum of sharp loss surface), and then the updated amount of new task $T+1$ is $\Delta\boldsymbol{W}$, and the weight of the network becomes $\boldsymbol{W}_2$. When we use $\boldsymbol{W}_2$ to predict the old task T, the loss for task T increases significantly (i.e., a large $F_{1,2}$). However, if the loss surface is flat enough, that is, task T converges to $\boldsymbol{W}_3$ (a minimum of flat loss surface), and task $T+1$ updated to $\boldsymbol{W}_4$, the loss of old task T does not change significantly (that is, $F_{3,4} << F_{1,2}$), i.e., old tasks are forgotten less.

Is the loss surface not flat in GPM? We are curious about whether the catastrophic forgetting of GPM at a small projection threshold is due to the loss surface is not flat. To answer this question, we quantitatively analyze the flatness of the loss surface in GPM in Fig. 4. Specifically, we use the maximum eigenvalue of the Hessian matrix w.r.t the weight $\boldsymbol{W}$ to measure the flatness of a loss surface, and a large eigenvalue means a sharp loss surface (The definition and measurement of flatness are explained in the appendix). We can observe that when the projection threshold ϵ_{th}^l de-

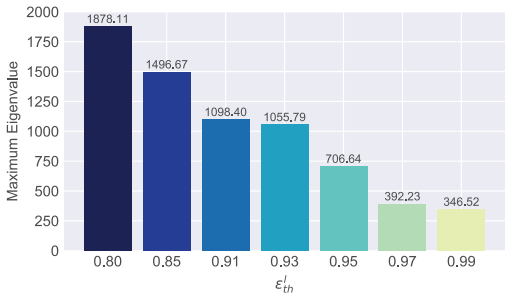

Figure 4: The maximum eigenvalue of task $T1$ in the GPM method under different projection thresholds on CIFAR-100.

creases, the loss surface of Task $T1$ becomes sharper, that is, the maximum eigenvalue increases. For example, when the projection threshold is 0.80 and 0.99 respectively, the maximum eigenvalue of the former is much higher than that of the latter, that is, $1878.11 >> 346.52$. Therefore, catastrophic forgetting is more likely to occur at a relatively small projection threshold due to a sharp loss surface in GPM.

Inspired by the above observations, we propose a novel gradient projection method for CL, namely data augmented flatness-aware gradient projection (DFGP). DFGP avoids catastrophic forgetting of old tasks by improving the flatness of the loss surface, so it allows a relatively small projection threshold to improve the performance of new tasks.

3. Methodology

In this section, we will first introduce the proposed DFGP method in Sec. 3.1 and then describe its components in detail in Sec. 3.2 and Sec. 3.3.

3.1. DFGP Formulation

Since GPM only minimizes empirical risk when learning task t, i.e., $\min_{\boldsymbol{W}} \mathcal{L}^t(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y})$, where $(\boldsymbol{X}, \boldsymbol{Y})$ is training data of task t (we omit the superscript for brevity), and does not consider any flatness-aware optimization objective, the loss surface in GPM is generally sharp. In this paper, inspired by sharpness-aware minimization (SAM) [16, 28, 48, 34], we optimize both the loss and the flatness of loss surface. The key of our method is to find the sharpest case of the loss surface from two levels of data and parameters to optimize to obtain a flat loss surface. Specifically, our proposed data augmented flatness-aware gradient projection (DFGP) overall optimization objective is as follows:

$$\begin{aligned}\min_{\boldsymbol{W}} \Big\{ & \mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \mathcal{L}^t\left(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right) \\ & + \max_{\|\boldsymbol{\delta}\|_2 \leq \rho} \Big[\left(\mathcal{L}^t\left(\boldsymbol{W}+\boldsymbol{\delta}, \boldsymbol{X}, \boldsymbol{Y}\right) - \mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right)\right) \\ & + \lambda\left(\mathcal{L}^t(\boldsymbol{W}+\boldsymbol{\delta}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}) - \mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})\right) \Big] \Big\},\end{aligned} \tag{2}$$

where $(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})$ represents the worst-case perturbation at the data level in Sec. 3.2(1), and $\boldsymbol{W}+\boldsymbol{\delta}$ represents the worst-case perturbation at the weight level in Sec. 3.2(2). The first row in Eq. 2 represents the loss minimization on the raw data $(\boldsymbol{X}, \boldsymbol{Y})$ and the worst-case perturbed data $(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})$, the second and third rows represent the flatness of loss surface on the original data and perturbed data. The goal of both data perturbation and weight perturbation is to find the worst-case of the network for optimization to obtain a flatness loss surface in Sec. 3.2(3). Compared with SAM [16], our method further performs data-level perturbation in addition to weight-level perturbation, which can better adapt to distribution drift between tasks and flatness-aware optimization on a wider exploration (data) space.

The algorithm for solving DFGP is summarized in Alg. 1. Below, we separately describe the solution process of data perturbation and weight perturbation to compute the flatness-aware gradient, respectively.

3.2. Data Augmented Flatness-aware Optimization

Our DFGP in Eq. 2 collaboratively improves the flatness of the loss surface by simultaneously disturbing the data and weight. Specifically, the goal of data perturbation is to find the worst-case perturbation $\hat{\gamma}$ that maximizes the loss $\mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}(\hat{\gamma}), \tilde{\boldsymbol{Y}}(\hat{\gamma}))$ for task t. The perturbed data $(\tilde{\boldsymbol{X}}(\hat{\boldsymbol{\gamma}}), \tilde{\boldsymbol{Y}}(\hat{\gamma}))$ are constructed by interpolating the raw data $(\boldsymbol{X}, \boldsymbol{Y})$ according to $\hat{\gamma}$. The goal of weight perturbation is to find a worst-case δ in the neighborhood of the weight $\boldsymbol{W}$, which maximizes the loss of task t. Afterward, DFGP optimizes the task loss and loss surface on the worst-case data and weight perturbations to obtain a flatness-aware gradient.

(1) Data perturbation. Inspired by Mixup [59, 60], we perturbed the data and extended the raw training data distribution. Specifically, we first interpolate any two samples $x_i^{(t)}$ and $x_j^{(t)}$ from $\boldsymbol{X}$ (and the corresponding labels $y_i^{(t)}$ and $y_j^{(t)}$ form $\boldsymbol{Y}$) in a minibatch from task t to generate a perturbed training example $(\tilde{x}^{(t)}(\gamma), \tilde{y}^{(t)}(\gamma))$: $\tilde{x}^{(t)}(\gamma) = \gamma x_i^{(t)} + (1-\gamma) x_j^{(t)}, \tilde{y}^{(t)}(\gamma) = \gamma y_i^{(t)} + (1-\gamma) y_j^{(t)}$, where $\gamma \in [0,1]$ represents the weight when the two samples are mixed. However, unlike the vanilla Mixup, which $\gamma \sim \text{Beta}(\alpha, \alpha)$ is directly sampled from a Beta distribution, we perform a further optimization on γ whose goal is to maximize the loss corresponding to the samples after interpolation to promote the flatness-aware optimization of the network. In other words, we will add a perturbation ϵ to the sampled γ in the neighborhood of radius ρ, so that the loss is worst-case on the new image mixed with the perturbed $\hat{\gamma}(\hat{\gamma} := \gamma + \epsilon)$. In addition, to ensure that the interpolated samples still belong to the class corresponding to $y_i^{(t)}$ and $y_j^{(t)}$, we clip $\hat{\gamma}$ to be between $[0, 1]$. The formal expression of our gamma's (i.e., $\hat{\gamma}$) goal is as follows:

$$\max_{(\hat{\gamma}:=\gamma+\epsilon)\in[0,1], \|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}^t\left(\boldsymbol{W}, \tilde{\boldsymbol{X}}(\gamma+\epsilon), \tilde{\boldsymbol{Y}}(\gamma+\epsilon)\right). \tag{3}$$

However, it is very difficult to exactly find the solution of

Algorithm 1 Algorithm of our DFGP

1: **Function** Train ($\boldsymbol{W}$, $\mathcal{D}_{train}$ η, ϵ_{th}, ε, ρ, λ, α)
2: Initialize: $\boldsymbol{M} \leftarrow \{(\boldsymbol{M}^l)_{l=1}^L\}$, $\boldsymbol{M}^l \leftarrow [\ ]$, for all $l = 1, 2, \ldots, L$
3: Initialize: $\boldsymbol{W} \leftarrow \boldsymbol{W}_0$
4: **for** task $t = 1, 2, \ldots, T$ **do**
5: **repeat**
6: $B_n^{(t)} \leftarrow (\boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)}) \sim \mathcal{D}_{train}^t$
7: $\boldsymbol{g}_{\boldsymbol{W}}^{(t)} \leftarrow$ FlatnessAwareGradient ($B_n^{(t)}$, $\boldsymbol{W}$, ρ, ε, λ, α)
8: **for** layer $l = 1, 2, \ldots, L$ **do**
9: **if** $t > 1$ **then**
10: $\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l} \leftarrow \boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l} - \text{Proj}(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}, \boldsymbol{M}^l)$
11: **end if**
12: $\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \cdot \boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}$
13: **end for**
14: **until** convergence
15: $\boldsymbol{M} \leftarrow$ UpdateGradientMemory ($\boldsymbol{W}$, $\boldsymbol{M}$, $\mathcal{D}_{train}^t$, ϵ_{th}, t)
16: **end for**
17: **return** $\boldsymbol{W}$
18: **end Function**
19:
20: **Procedure** FlatnessAwareGradient (B_n, $\boldsymbol{W}$, ρ, ε, λ, α)
21: $B_n' \leftarrow (\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}) \leftarrow \text{Mixup}(B_n, \gamma)$, $\gamma' \leftarrow \text{Beta}(\alpha, \alpha)$
22: $\hat{\epsilon} \leftarrow \rho \cdot \frac{\nabla_\gamma \mathcal{L}^t(B_n')}{\|\nabla_\gamma \mathcal{L}^t(B_n')\| + \varepsilon}$
23: $\hat{\boldsymbol{\delta}} \leftarrow \rho \cdot \frac{\nabla_{\boldsymbol{W}} \mathcal{L}^t(B_n) + \lambda \nabla_{\boldsymbol{W}} \mathcal{L}^t(B_n')}{\|\nabla_{\boldsymbol{W}} \mathcal{L}^t(B_n) + \lambda \nabla_{\boldsymbol{W}} \mathcal{L}^t(B_n')\| + \varepsilon}$
24: $\hat{\gamma} \leftarrow \text{Clamp}(\gamma + \hat{\epsilon}, 0, 1)$, $\hat{\boldsymbol{W}} \leftarrow \boldsymbol{W} + \hat{\boldsymbol{\delta}}$
25: $B_n'' \leftarrow (\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}) \leftarrow \text{Mixup}(B_n, \hat{\gamma})$
26: $\boldsymbol{g}_{\boldsymbol{W}}^{(t)} \leftarrow \nabla_{\boldsymbol{W}}(\mathcal{L}^t(B_n) + \lambda \mathcal{L}^t(B_n''))|_{\hat{\boldsymbol{W}}}$
27: **return** $\boldsymbol{g}_{\boldsymbol{W}}^{(t)}$, where $\boldsymbol{g}_{\boldsymbol{W}}^{(t)} = \{(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t)})_{l=1}^L\}$
28: **end Procedure**
29:
30: **Procedure** UpdateGradientMemory ($\boldsymbol{W}$, $\boldsymbol{M}$, $\mathcal{D}_{train}^t$, ϵ_{th}, t)
31: $B_n^{(t)} \leftarrow (\boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)}) \sim \mathcal{D}_{train}^t$
32: $\boldsymbol{R}^{(t)} \leftarrow \text{forward}(B_n^{(t)}, \boldsymbol{W})$, where $\boldsymbol{R}^{(t)} = \{(\boldsymbol{R}^{(t),l})_{l=1}^L\}$
33: **for** layer $l = 1, 2, \ldots, L$ **do**
34: $\hat{\boldsymbol{R}}^{(t),l} = \boldsymbol{R}^{(t),l}$
35: **if** $t > 1$ **then**
36: $\hat{\boldsymbol{R}}^{(t),l} = \boldsymbol{R}^{(t),l} - \text{Proj}_{S^l}(\boldsymbol{R}^{(t),l})$
37: **end if**
38: $\hat{\boldsymbol{U}}^{(t),l} \leftarrow \text{SVD}(\hat{\boldsymbol{R}}^{(t),l})$
39: $k \leftarrow \text{criteria}(\hat{\boldsymbol{R}}^{(t),l}, \boldsymbol{R}^{(t),l}, \epsilon_{th}^l)$
40: $\boldsymbol{M}^l \leftarrow [\boldsymbol{M}^l, \hat{\boldsymbol{U}}^{(t),l}[0:k]]$
41: **end for**
42: **return** $\boldsymbol{M}$
43: **end Procedure**

Eq. 3, so the optimization goal of ϵ is the following first-order Taylor approximate problem, and the solution $\hat{\epsilon}$ is:

$$\begin{aligned}\hat{\epsilon} \approx & \underset{\|\boldsymbol{\epsilon}\|_2 \leq \rho}{\arg\max}\ \mathcal{L}^t\left(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right) + \boldsymbol{\epsilon}^\top (\nabla_\gamma \mathcal{L}^t\left(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right) \\ = & \rho \cdot \nabla_\gamma \mathcal{L}^t\left(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right) / \|\nabla_\gamma \mathcal{L}^t\left(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right)\|_2\end{aligned} \tag{4}$$

(2) Weight perturbation. At the same time, we perform weight perturbation on raw data $(\boldsymbol{X}, \boldsymbol{Y})$ and Mixup data $(\tilde{\boldsymbol{X}}(\gamma), \tilde{\boldsymbol{Y}}(\gamma))$ for task t. Note that the reason the weight perturbation is also performed on the augmented data is to allow the weight perturbation to explore in a wider data space. Specifically, the objective of the weight perturbation is to find the $\boldsymbol{\delta}$ that causes the worst-case loss of task t in the neighborhood with $\boldsymbol{W}$ as the center and ρ as the radius, i.e.,

$$\max_{\|\boldsymbol{\delta}\|_2 \leq \rho} \mathcal{L}^t\left(\boldsymbol{W} + \boldsymbol{\delta}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \mathcal{L}^t\left(\boldsymbol{W} + \boldsymbol{\delta}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right). \tag{5}$$

We obtain the solution of $\hat{\boldsymbol{\delta}}$ by solving the first-order Taylor approximation problem of problem Eq. 5 as follows:

$$\begin{aligned}\hat{\boldsymbol{\delta}} \approx & \underset{\|\boldsymbol{\delta}\|_2 \leq \rho}{\arg\max}\ \mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}) \\ & + \boldsymbol{\delta}^\top \left(\nabla_{\boldsymbol{W}} \mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \nabla_{\boldsymbol{W}} \mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})\right) \\ = & \rho \cdot \frac{\nabla_{\boldsymbol{W}} \mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \nabla_{\boldsymbol{W}} \mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})}{\left\|\nabla_{\boldsymbol{W}} \mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \nabla_{\boldsymbol{W}} \mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})\right\|_2}.\end{aligned} \tag{6}$$

It should be mentioned that, observing the forms of the two optimization problems of Eq. 6 and Eq. 4, we can find that the optimal weight perturbation $\hat{\boldsymbol{\delta}}$ and data perturbation $\hat{\epsilon}$ can be computed together through a single backpropagation, as opposed to requiring separate backpropagations.

(3) Flatness-aware gradient optimization. The flatness-aware optimization problem is to optimize the loss in Eq. 2 on the worst-case data and weight perturbation. When Eq. 2 is minimized, the model converges to a minimum $\boldsymbol{W}^*$ of a flat loss surface. Bringing the worst-case perturbed data $(\hat{\boldsymbol{X}}(\hat{\gamma}), \hat{\boldsymbol{Y}}(\hat{\gamma}))$ and weight $\hat{\boldsymbol{W}}$ into Eq. 2 and removing redundant terms, our flatness-aware minimization objective function is:

$$\min_{\boldsymbol{W}} \mathcal{L}^t\left(\hat{\boldsymbol{W}}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \mathcal{L}^t\left(\hat{\boldsymbol{W}}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right), \tag{7}$$

where $\hat{\gamma} = \gamma + \hat{\epsilon}$, $\hat{\boldsymbol{W}} = \boldsymbol{W} + \hat{\boldsymbol{\delta}}$; $\hat{\epsilon}$ and $\hat{\boldsymbol{\delta}}$ are from Eq. 4 and Eq. 6, respectively. Therefore, we compute the final flatness-aware gradient approximation to update parameters $\boldsymbol{W}$ as:

$$\boldsymbol{g}_{\boldsymbol{W}}^{(t)} \approx \nabla_{\boldsymbol{W}}(\mathcal{L}^t\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \lambda \mathcal{L}^t(\boldsymbol{W}, \tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}))\Big|_{\hat{\boldsymbol{W}}}. \tag{8}$$

So far, we have obtained a flatness-aware gradient (i.e., $\boldsymbol{g}_{\boldsymbol{W}}^{(t)}$ in Eq. 8) with respect to $\boldsymbol{W}$ that optimizes both loss and flatness of loss surface for task t.

3.3. Flatness-aware Gradient Projection

In this subsection, we describe how to combine the gradient of data augumented flatness-aware optimization with the orthogonal gradient projection method in CL.

Learning task 1. The learning process of task 1 includes gradient update and projected memory update. *1) Weight update:* The first task in CL is learned without any constraints. Therefore, after obtaining the gradient in Eq. 8 (see line 7 in Alg. 1), we directly perform the following gradient descent for parameter update of layer l (see line 12 in Alg. 1): $\boldsymbol{W}^l = \boldsymbol{W}^l - \eta \cdot \boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}, l \in \{1, \ldots, L\}$, where

η is the step size. *2) Memory update:* After the task 1 has converged, we need to calculate the core gradient space (CGS) of task 1 to construct the projection matrix $\boldsymbol{M}$. We first select randomly n samples to obtain the representation $\boldsymbol{R}^{(1),l}=[x_1^{(1),l},\ldots,x_n^{(1),l}]$ of each layer $l\in\{1,\ldots,L\}$ (see line 32 in Alg. 1), and then perform SVD decomposition on this representation $\boldsymbol{R}^{(1),l}=\boldsymbol{U}^{(1),l}\boldsymbol{\Sigma}^{(1),l}\boldsymbol{V}^{(1),l}$ to select the top-k most important basis for task 1 (see lines 38-39 in Alg. 1). Finally, we take the eigenvectors corresponding to the top-k maximum eigenvalues as the core gradient space of task 1, that is, $\boldsymbol{M}^l=span\{u_1^{(1),l},\ldots,u_k^{(1),l}\}$ (see line 40 in Alg. 1). When task 2 updates the parameters, it needs to be orthogonal to this core gradient space, which can effectively reduce the interference to task 1.

Learning task 2 to T. The learning process of task $2\sim T$ includes gradient projection, gradient update, and projected memory update. *1) Gradient Projection:* Similar to task 1, we use Eq. 8 to compute the flatness-aware gradient $\boldsymbol{g}_{\boldsymbol{W}}^{(t)}$ for task t (see line 7 in Alg. 1). But instead of using the gradient directly for updating, the gradient components parallel to task $t-1$ are eliminated (i.e., $\text{Proj}_{S^l}(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l})=\boldsymbol{M}^l(\boldsymbol{M}^l)^\top\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}$), and only the gradient components orthogonal to task $t-1$ are retained (i.e., $\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}$–$\text{Proj}_{S^l}(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l})$). *2) Weight update:* We update the parameters with the remaining gradient (see lines 9-12 in Alg. 1), that is: $\boldsymbol{W}^l=\boldsymbol{W}^l-\eta\cdot(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}-\text{Proj}_{S^l}(\boldsymbol{g}_{\boldsymbol{W}^l}^{(t),l}))$. *3) Memory update:* Since the space of previous $t-1$ tasks may contain significant gradient directions of the task t, when building the core gradient space of task t, only the important basis additionally included in task t is added to the gradient memory $\boldsymbol{M}^l$. Speficically, we first project the representation $\boldsymbol{R}^{(t),l}=[x_1^{(t),l},\ldots,x_n^{(t),l}]$ of task t for each layer l into the gradient space formed by previous tasks to remove duplicate representations (see line 36 in Alg. 1): $\hat{\boldsymbol{R}}^{(t),l}=\boldsymbol{R}^{(t),l}-\text{Proj}_{S^l}(\boldsymbol{R}^{(t),l})=\boldsymbol{R}^{(t),l}-\boldsymbol{M}^l(\boldsymbol{M}^l)^\top\boldsymbol{R}^{(t),l}$. Then we perform SVD decomposition ($\hat{\boldsymbol{R}}^{(t),l}=\hat{\boldsymbol{U}}^{(t),l}\hat{\boldsymbol{\Sigma}}^{(t),l}(\hat{\boldsymbol{V}}^{(t),l})$) on this part of the representation and require that the following formula be satisfied (see lines 38-39 in Alg. 1): $\|\text{Proj}(\boldsymbol{R}^{(t),l})\|_F^2+\|(\hat{\boldsymbol{R}}_k^{(t),l})\|_F^2\geq\epsilon_{th}^l\|\boldsymbol{R}^{(t),l}\|_F^2$. Finally, we add the eigenvectors corresponding to the maximum top-k eigenvalues of task t into the gradient memory, i.e., $\boldsymbol{M}^l\leftarrow[\boldsymbol{M}^l,\hat{\boldsymbol{U}}^{(t),l}[0:k]]$ (see line 40 in Alg. 1).

4. Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness of DGFP. The implementation details and additional experiments are provided Appendix.

4.1. Experimental Setup

Datasets We evaluate our method on four benchmark datasets for CL [43, 27]: Permuted MNIST, 10-Split CIFAR-100, 5-Datasets and 20-Spilt miniImageNet.

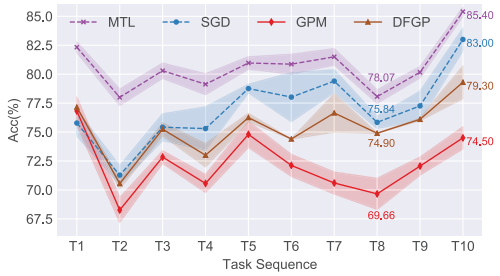

Figure 5: New tasks' accuracy of GPM, SGD, MTL, and DFGP on CIFAR-100.

Baselines. We compare DFGP with four types of CL methods. Architecture-based method: HAT [44]. Regularization-based methods: EWC [23], MAS [1]. Memory-based methods: ER [10], A-GEM [9]. Orthogonal-Projection-based methods: OWM [57], GPM [43], FS-DGPM [12]. We also combine GPM with Classifier-Projection Regularization(CPR) [6] as a strong baseline (GPM+CPR).

Evaluation metrics. We evaluate the performance by average accuracy (ACC) and backward transfer (BWT) [43]. Specifically, ACC represents the *average test accuracy* of the model trained on all tasks. BWT measures the *forgetting* of old tasks. ACC and BWT are defined as: $\text{ACC}=\frac{1}{T}\sum_{i=1}^{T}A_{T,i}$, $\text{BWT}=\frac{1}{T-1}\sum_{i=1}^{T-1}A_{T,i}-A_{i,i}$, where $A_{t,i}$ is the accuracy of the model tested on task i after the training of task t is completed. T is the number of tasks.

4.2. Experimental Results

Performance. As shown in Tab. 1, the accuracy of DFGP is significantly improved over previous work on all datasets. For example, DFGP achieves the accuracy gains of 1.07%, 1.01%, 0.77%, and 7.78% on the four datasets PMNIST, CIFAR-100, 5-Datasets, and MiniImageNet, respectively, compared to the best baseline methods GPM+CPR, FS-DGPM, HAT, and GPM+CPR. It should be mentioned that these datasets are widely studied benchmark datasets in CL. Therefore, the improvement of our method on these datasets is already significant. In addition, there is a clear trend that DFGP can achieve more significant benefits compared with other methods with a larger number of tasks. For example, the number of tasks contained in PMNIST, CIFAR-100, 5-Datasets, and MiniImageNet are 10, 10, 5, and 20, respectively, so the accuracy gain shows a trend of MiniImageNet $>$ PMNIST, CIFAR-100 $>$ 5-Datasets. This is because old tasks are more likely to be forgotten when the number of tasks increases. DFGP method obviously alleviates the forgetting problem, and thus achieves a more significant improvement on the dataset with a larger number of tasks. Specifically, DFGP shows the lowest forgetting degree (BWT in Tab. 1) compared to other competitive methods.

Stability-Plasticity analysis. As shown in Fig. 6(a-b), when we reduce ϵ_{th}^l (i.e., our DFGP is $\epsilon_{th}^l=0.95$, GPM is $\epsilon_{th}^l=0.97$), DFGP achieves comparable or even better

Table 1: The averaged accuracy (ACC) and backward transfer (BWT) over all tasks on different datasets. Note that, MTL learns all tasks simultaneously in a single network by using the entire dataset, which does not belong to the setting of CL, it can be used as an upper bound for CL learning. SGD means that new tasks are learned without any constraints.

Method	PMNIST (10 Tasks)		CIFAR-100 (10 Tasks)		5-Datasets (5 Tasks)		MiniImageNet (20 Tasks)	
	ACC(%)	BWT	ACC(%)	BWT	ACC(%)	BWT	ACC(%)	BWT
SGD	53.56±2.89	-0.48±0.03	56.39±1.14	-0.22±0.01	80.23±1.16	-0.16±0.01	49.71±1.80	-0.21±0.01
MTL	96.70±0.02	-0.00±0.00	80.67±0.42	-0.00±0.00	93.15±0.16	-0.00±0.00	84.16±1.26	-0.00±0.00
EWC	89.97±0.57	-0.04±0.01	68.80±0.88	-0.02±0.01	88.64±0.26	-0.04±0.01	52.01±2.53	-0.12±0.03
MAS	86.98±0.84	-0.04±0.01	67.96±0.44	-0.05±0.00	88.73±0.79	-0.04±0.01	60.80±2.96	-0.09±0.02
HAT	-	-	72.06±0.50	-0.00±0.00	91.32±0.18	-0.01±0.00	59.78±0.57	-0.03±0.00
A-GEM	83.56±0.16	-0.14±0.00	63.98±1.22	-0.15±0.02	84.04±0.33	-0.12±0.01	57.24±0.72	-0.12±0.01
ER	87.24±0.53	-0.11±0.01	71.73±0.63	-0.06±0.01	88.31±0.22	-0.04±0.00	58.94±0.85	-0.07±0.01
OWM	90.71±0.11	-0.01±0.00	50.94±0.60	-0.30±0.01	-	-	-	-
GPM	93.18±0.14	-0.03±0.00	72.31±0.20	-0.00±0.00	91.12±0.00	-0.01±0.00	60.99±2.01	-0.05±0.01
FS-DGPM	92.89±0.18	-0.36±0.01	73.58±0.14	-0.30±0.00	-	-	-	-
GPM+CPR	93.57±0.15	-0.03±0.00	72.21±0.43	-0.00±0.00	89.72±0.48	-0.01±0.00	62.14±1.89	-0.04±0.01
DFGP	**94.64±0.17**	-0.01±0.00	**74.59±0.33**	-0.00±0.00	**92.09±0.18**	-0.01±0.00	**69.92±0.90**	-0.01±0.00

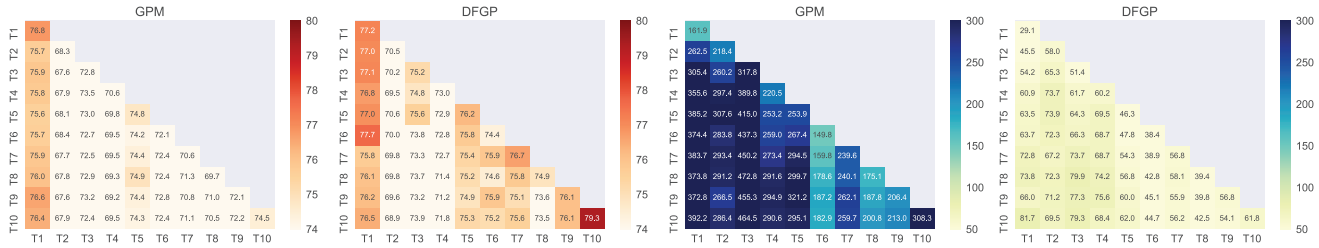

Figure 6: The accuracy (Higher Better) and the maximum eigenvalue (Lower Better) on the CIFAR-100 dataset. (a) Acc of GPM: best $\epsilon_{th}^{l}=0.97$; (b) Acc of our DFGP: best $\epsilon_{th}^{l}=0.95$; (c) Maximum eigenvalue of GPM; (d) Maximum eigenvalue of our DFGP. t-th row represents the accuracy of the network tested on tasks $1-t$ after task t is learned.

than GPM on the old tasks. For example, when task $T10$ is learned, the accuracy of GPM on tasks $T2$, $T4$, and $T6$ is 67.9%, 69.5%, and 72.4%, respectively. DFGP is 68.9%, 71.8%, and 75.2%, respectively. These evidences suggest that our method flattening the loss surface really helps alleviate catastrophic forgetting. In addition, we highlight the improved performance that our method brings to new tasks. As shown in Fig. 5, GPM can only achieve the accuracy of 69.66% and 74.50% when learning new task $T8$ and new task $T10$, while DFGP achieves the accuracy of 74.90% and 79.30% respectively. In conclusion, our method greatly improves the performance of new tasks while maintaining that old tasks are not catastrophically forgotten.

4.3. Ablation Study

Effectiveness of each component. Compared with GPM, DFGP perturbs both data and weight to perform flatness-aware optimization. As shown in Tab. 2, when we perform perturbation on only one level, it suffers from a certain accuracy drop, proving the effectiveness of both components in DFGP. For example, when using only weight perturbation (w/ $\mathrm{W}(\hat{\delta})$) or data perturbation (w/ $\mathrm{D}(\hat{\gamma})$) on CIFAR-100, the ACC drops from 74.59% to 73.46% and 73.19%. After we replace the flatness-aware data perturbation (w/ $\mathrm{D}(\hat{\gamma})$) with vanilla Mixup (w/ $\mathrm{D}(\gamma)$), the accuracy will be further reduced. For example, on PMNIST(CIFAR-100), it has dropped from 94.04%(73.19%) to 93.81%(72.94%).

Table 2: Effectiveness of each component.

Method	PMNIST		CIFAR-100	
	ACC(%)	BWT	ACC(%)	BWT
GPM	93.18	-0.03	72.31	-0.00
w/ $\mathrm{W}(\hat{\delta})$	94.49	-0.02	73.46	-0.01
w/ $\mathrm{D}(\hat{\gamma})$	94.04	-0.03	73.19	-0.02
w/ $\mathrm{D}(\gamma)$	93.81	-0.03	72.94	-0.02
DFGP	94.64	-0.01	74.59	-0.00

Flatness visualization. We verify that DFGP shows a flatter loss surface compared to GPM. We measure the flatness of loss surface via the maximum eigenvalue of the network's Hessian matrix in Fig. 6(c-d). The maximum eigenvalue corresponding to DFGP is much smaller than that of GPM, e.g., the maximum eigenvalues of the former and the latter in task $T10$ are 61.8 and 308.3, respectively.

Combined with other CL methods. We further combine Data augmented Flatness-aware optimization (abbreviated as DF) with other three kinds of CL algorithms, including orthogonal projection based (TRGP [27]), memory-based (ER [10]), and regularization-based (MAS [1]) methods.

Table 3: Data augmented flatness-aware optimization on more CL methods for accuracy and BWT testing.

Method	PMNIST		CIFAR-100	
	ACC(%)	BWT	ACC(%)	BWT
TRGP	96.34	-0.01	72.67	-0.19
DF-TRGP (ours)	**96.90**	-0.01	**73.63**	-0.01
ER	87.24	-0.11	71.73	-0.06
DF-ER (ours)	**89.99**	-0.08	**73.62**	-0.07
MAS	86.98	-0.04	67.96	-0.05
DF-MAS (ours)	**88.90**	-0.03	**70.44**	-0.02

Table 4: The averaged accuracy (ACC) of GPM and DFGP when tested on adversarial examples $(\boldsymbol{X}^{adv}, \boldsymbol{Y})$.

	PMNIST		CIFAR-100	
	GPM	DFGP	GPM	DFGP
μ=0.0	93.56(-0.00)	**94.64**(-0.00)	72.31(-0.00)	**74.59**(-0.00)
μ=0.0001	93.51(-0.05)	**94.60**(-0.04)	71.91(-0.40)	**74.50**(-0.09)
μ=0.001	93.07(-0.49)	**94.31**(-0.34)	71.43(-0.88)	**73.92**(-0.67)
μ=0.01	87.81(-5.75)	**90.80**(-4.05)	66.33(-5.97)	**68.91**(-5.68)

Specifically, we use the flatness-aware gradient proposed in this paper instead of the vanilla gradient for parameter updates of three approaches. As shown in Tab. 3, on the PMNIST, DF-ER and DF-MAS achieve absolute accuracy gains of 2.75% and 1.92% over ER and MAS, respectively. On the CIFAR-100, DF-TRGP, DF-ER and DF-MAS achieve absolute accuracy gains of 0.96%, 1.89% and 2.48% over TRGP, ER and MAS, respectively.

Robustness analysis. We verify that DFGP is more adversarial robust than GPM. To test the robustness, we refer to the popular FGSM [18] method for adversarial perturbation of the input image, and the detailed attack rule is: $\boldsymbol{X}^{adv} = \boldsymbol{X} + \mu \cdot \text{sign}\left(\nabla_{\boldsymbol{X}} \mathcal{L}^t(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y})\right)$, where μ is a hyperparameter of adversarial strength, and $\boldsymbol{X}$ represents the samples of the test set. As shown in Tab. 4, we find that the larger μ is, the greater the performance degrades of both methods. In addition, the performance drop of DFGP on both datasets is smaller than that of GPM. For example, when μ is 0.01, on CIFAR100 dataset, the former and the latter decrease by -5.97 and -5.68; on PMNIST dataset, GPM and DFGP decrease by -5.75 and -4.05, respectively.

5. Related Work

Continual learning. CL can be divided into two types: Online CL [3, 24] models process the data in a single pass, while offline CL models process the data in multiple passes. This paper mainly focuses on offline CL, we leave online CL for future work. Existing offline CL methods can be roughly divided into four categories: 1) *Architecture-based methods* [44, 31, 42, 35] add a new set of learnable parameters for each new task. However, these approaches cause the number of parameters to grow linearly with the number of tasks. 2) *Regularization-based methods* [23, 6, 58, 2, 7, 30] calculate the importance of network parameters for each old task and reduce the updating of important parameters for the old tasks. However, these methods need to store the importance of each parameter for the old tasks, which results in huge memory overhead. In addition, calculating the importance of parameters also requires a time cost. 3) *Memory-based methods* [10, 9, 41, 46, 52] use a fixed memory to store a small number of training samples from old tasks and replay these samples when learning new tasks. However, in some privacy-strict cases, caching of arbitrary historical samples is not allowed. 4) *Orthogonal projection based methods* [57, 15, 12, 27, 25, 43] restrict the gradient directions of new tasks to alleviate interference with old tasks. In particular, Gradient Projection Memory (GPM) [43] performs better than other methods.

Flatness in deep learning. Recently, many machine learning works have attempted to improve the flatness of the loss surface by sharpness-aware minimization [22, 16, 4, 48, 34]. Stochastic weight average [21, 54] is also a approach for achieving flatness. Flatness has also been explored in the CL. StableSGD [36] empirically analyzes how the learning rate, mini-batch size, and dropout affect the flatness in CL. [32] focuses on explaining the effect of pre-training on improving the performance of CL from the perspective of flatness. CPR [6] improves model generalization by maximizing the entropy of classifier output probabilities [39]. Whereas our DFGP directly optimizes the flatness of loss surface. F2M improves flatness when training basic tasks in incremental few-shot learning scenarios [45]. Unlike F2M, our DFGP focuses on the standard CL scenario. A related work is FS-DGPM [12], which scales the basis in GPM and performs flat optimization with adversarial weight perturbations [55]. However, our method is fundamentally different from FS-DGPM: 1) Our framework is more general. We introduce flatness-aware optimization from both data and weight levels, while FS-DGPM merely considers the weight level. 2) Our method is substantially more efficient and effective than FS-DGPM. We solve weight perturbation by first-order Taylor approximation, while FS-DGPM uses gradient ascent iteration. 3) We provide a deeper insight into how flatness affects the performance of old and new tasks in GPM, i.e., we establish a connection between flatness and projection criteria in GPM, while FS-DGPM ignores this relation.

6. Conclusion and Future Work

In this work, we first revisit the problem of poor performance for new tasks in GPM from a flat loss surface perspective. Then, we propose a data augmented flatness-aware gradient projection (named DFGP) algorithm to improve the flatness of loss surface. Our proposed DFGP helps to maintain the stability of old tasks and improve the performance of new tasks. Next, we demonstrate that the proposed

data augmented flatness-aware optimization strategy can also be combined with other CL methods to help alleviate catastrophic forgetting. Finally, extensive experiments demonstrate that our proposed DFGP can help to improve flatness and robustness for the CL problem. In the future, we intend to directly use the degree of flatness to guide the design of projection criteria for each layer of the CL network. In addition, we also intend to explore the effectiveness of flatness-aware optimization for online CL.

Acknowledgements

Enneng Yang, Guibing Guo are supported by the National Natural Science Foundation of China under Grants No. 62032013 and No. 61972078, and the Fundamental Research Funds for the Central Universities under Grants No. N2217004 and No. N2317002. Li Shen is supported by the STI 2030—Major Projects (No. 2021ZD0201405).

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, volume 11207, pages 144–161, 2018. 6, 7, 11, 12, 13

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, volume 11207, pages 144–161, 2018. 8

[3] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *NeurIPS*, pages 11816–11825, 2019. 8

[4] Maksym Andriushchenko and Nicolas Flammarion. Towards understanding sharpness-aware minimization. In *ICML*, volume 162, pages 639–668, 2022. 8

[5] Yaroslav Bulatov. Notmnist dataset. *Google (Books/OCR), Tech. Rep.[Online]*, 2, 2011. 11

[6] Sungmin Cha, Hsiang Hsu, Taebaek Hwang, Flávio P. Calmon, and Taesup Moon. CPR: classifier-projection regularization for continual learning. In *ICLR*, 2021. 6, 8, 12

[7] Arslan Chaudhry, Puneet Kumar Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, volume 11215, pages 556–572, 2018. 8

[8] Arslan Chaudhry, Naeemullah Khan, Puneet K. Dokania, and Philip H. S. Torr. Continual learning in low-rank orthogonal subspaces. In *NeurIPS*, 2020. 1

[9] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *ICLR*, 2019. 6, 8, 11

[10] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet Kumar Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. *CoRR*, abs/1902.10486, 2019. 6, 7, 8, 11, 13

[11] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020. 3

[12] Danruo Deng, Guangyong Chen, Jianye Hao, Qiong Wang, and Pheng-Ann Heng. Flattening sharpness for dynamic gradient projection memory benefits continual learning. In *NeurIPS*, pages 18710–18721, 2021. 6, 8, 12, 13

[13] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. In *ICLR*, 2020. 11

[14] Sayna Ebrahimi, Franziska Meier, Roberto Calandra, Trevor Darrell, and Marcus Rohrbach. Adversarial continual learning. In *ECCV*, volume 12356, pages 386–402, 2020. 11

[15] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *AISTATS*, volume 108, pages 3762–3773, 2020. 1, 3, 8

[16] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *ICLR*, 2021. 3, 4, 8, 13, 14

[17] Robert M. French. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In *NeurIPS*, pages 1176–1177, 1993. 1

[18] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015. 8

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 11

[20] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997. 2, 3, 13

[21] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, pages 876–885, 2018. 8

[22] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017. 3, 8, 13

[23] 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. 6, 8, 11

[24] Hyunseo Koh, Dahyun Kim, Jung-Woo Ha, and Jonghyun Choi. Online continual learning on class incremental blurry task configuration with anytime inference. In *ICLR*, 2022. 8

[25] Yajing Kong, Liu Liu, Zhen Wang, and Dacheng Tao. Balancing stability and plasticity through advanced null space in continual learning. In *ECCV*, pages 219–236. Springer, 2022. 8

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1106–1114, 2012. 11

[27] Sen Lin, Li Yang, Deliang Fan, and Junshan Zhang. TRGP: trust region gradient projection for continual learning. In *ICLR*, 2022. 1, 3, 6, 7, 8, 13

[28] Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Towards efficient and scalable sharpness-aware minimization. In *CVPR*, pages 12350–12360, 2022. 4, 14

[29] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, pages 6467–6476, 2017. 11

[30] Divyam Madaan, Jaehong Yoon, Yuanchun Li, Yunxin Liu, and Sung Ju Hwang. Representational continuity for unsupervised continual learning. In *ICLR*, 2022. 8

[31] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, pages 7765–7773, 2018. 8

[32] Sanket Vaibhav Mehta, Darshan Patil, Sarath Chandar, and Emma Strubell. An empirical investigation of the role of pre-training in lifelong learning. *Arxiv*, 2021. 8

[33] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects, 2013. 1

[34] Peng Mi, Li Shen, Tianhe Ren, Yiyi Zhou, Xiaoshuai Sun, Rongrong Ji, and Dacheng Tao. Make sharpness-aware minimization stronger: A sparsified perturbation approach. *NeurIPS*, 35:30950–30962, 2022. 4, 8

[35] Zichen Miao, Ze Wang, Wei Chen, and Qiang Qiu. Continual learning with filter atom swapping. In *ICLR 2022*, 2022. 8

[36] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. In *NeurIPS*, 2020. 8

[37] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 11

[38] German Ignacio 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. 1, 2

[39] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR, Workshop Track Proceedings*, 2017. 8, 12

[40] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990. 1

[41] 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. In *ICLR*, 2019. 8

[42] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. 8

[43] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *ICLR*, 2021. 1, 3, 6, 8, 11, 12

[44] Joan Serrà, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, volume 80, pages 4555–4564, 2018. 6, 8, 11

[45] Guangyuan Shi, Jiaxin Chen, Wenlong Zhang, Li-Ming Zhan, and Xiao-Ming Wu. Overcoming catastrophic forgetting in incremental few-shot learning by finding flat minima. In *NeurIPS*, pages 6747–6761, 2021. 8

[46] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NeurIPS*, pages 2990–2999, 2017. 8

[47] Shagun Sodhani, Mojtaba Faramarzi, Sanket Vaibhav Mehta, Pranshu Malviya, Mohamed Abdelsalam, Janarthanan Rajendran, and Sarath Chandar. An introduction to lifelong supervised learning. *CoRR*, abs/2207.04354, 2022. 1, 2

[48] Hao Sun, Li Shen, Qihuang Zhong, Liang Ding, Shixiang Chen, Jingwei Sun, Jing Li, Guangzhong Sun, and Dacheng Tao. Adasam: Boosting sharpness-aware minimization with adaptive learning rate and momentum for training deep neural networks. *arXiv preprint arXiv:2303.00565*, 2023. 4, 8

[49] Vladimir Vapnik. *Statistical learning theory*. Wiley, 1998. 2

[50] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NeurIPS*, pages 3630–3638, 2016. 11

[51] Zhenyi Wang, Li Shen, Tiehang Duan, Donglin Zhan, Le Fang, and Mingchen Gao. Learning to learn and remember super long multi-domain task sequence. In *CVPR*, pages 7982–7992, 2022. 1

[52] Zhenyi Wang, Li Shen, Le Fang, Qiuling Suo, Tiehang Duan, and Mingchen Gao. Improving task-free continual learning by distributionally robust memory evolution. In *ICML*, pages 22985–22998. PMLR, 2022. 8

[53] Zhenyi Wang, Enneng Yang, Li Shen, and Heng Huang. A comprehensive survey of forgetting in deep learning beyond continual learning. *arXiv preprint arXiv:2307.09218*, 2023. 1

[54] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, volume 162, pages 23965–23998, 2022. 8

[55] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. *NeurIPS*, 33:2958–2969, 2020. 8, 12, 14

[56] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. 11

[57] Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019. 1, 3, 6, 8, 12

[58] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, volume 70, pages 3987–3995, 2017. 8

[59] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*. OpenReview.net, 2018. 4

[60] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? In *ICLR*, 2021. 4