Appendix overview. In this appendix, we provide details omitted in the main text, including:

- Experimental details in Appendix A: This section provides dataset statistics, architecture details, baseline introduction, implementation details, and training details.
- Additional definition and experimental results in Appendix B: This section provides the definition of measuring the flatness of the loss surface and supplements the experimental results on more datasets.

A. Experimental Details

Dataset description and statistics. We evaluate our method on PMNIST, CIFAR-100, 5-Datasets and miniImageNet datasets. 1) Permuted MNIST (PMNIST): We follow GEM [29], UCB [13] and GPM [43] to generate 10 sequence tasks with 10 different permutations of the MNIST dataset, each containing 10 classes of handwritten digit classification. 2) 10-Split CIFAR-100: We divide the CIFAR-100 dataset into 10 tasks, each containing 10 classes. 3) 5-Datasets: We follow ACL [14] and GPM, and take the following five dataset sequences as the tasks to be learned: CIFAR-10, MNIST, SVHN [37], not-MNIST [5] and FashionMNIST [56]. 4) 20-Spilt miniImageNet: Similarly to GPM, we divide the 100 classes in ImageNet [50] into 20 sequentially arriving learning tasks, each containing 5 classes.

Tab. 5 lists the dataset statistics used in this paper. Specifically, #Tasks indicates how many sequentially arrived tasks the dataset contains. #Classes indicates how many categories/classes each task contains. #Train, #Valid, and #Test respectively represent the number of samples of the training set, validation set, and test set for each task.

Table 5: Statistics of the datasets.

Datasets	#Tasks	#Classes	#Train	#Valid	#Test
PMNIST	10	10	54,000	6,000	10,000
CIFAR-100	10	10	4,750	250	1,000
MiniImageNet	20	5	2,450	50	500
CIFAR-10	5	10	47,500	2,500	10,000
MNIST		10	57,000	3,000	10,000
SVHN		10	69,595	3,662	26,032
Fashion MNIST		10	57,000	3,000	10,000
NotMNIST		10	16,011	842	1,873

Architecture details. Since all tasks are classified tasks, we use cross entropy plus softmax as the prediction probability for the output layer of all networks. In addition, all network layers use Relu as the activation function. Detailed configuration information for each network architecture is described below. *1) MLP:* The PMNIST dataset uses a MLP architecture. Like GPM [43], this architecture contains two hidden layers, each containing 100 neurons, without bias. *2) AlexNet:* The CIFAR-100 dataset uses an AlexNet architecture. Similar to GPM, AlexNet [26] consists of three convolutional layers and two fully connected layers. Specifically, the number of channels in the three convolution layers is respectively 64, 128, 256, and the convolution kernel size is respectively 4×4, 3×3, 2×2. The number of neurons in the two fully connected layers is 2048. We use Dropout to avoid overfitting, and the drop probability of the first two convolution layers is set to 0.2. The drop probability of the third convolution layer and the two full connection layers is set to 0.5. In addition, all layers use batch normalization. *3) ResNet18:* Following GPM and GEM [29], the 5-Datasets and MiniImageNet datasets used a smaller version of the ResNet18 [19] architecture. Specifically, the first seventeen layers are the convolution layer, and the last is a fully connected layer. The convolution kernel size and padding size of all convolution layers are set to 3×3 and 1×1, respectively. The stride of the 1, 6, 10, and 14-th layers is set to 2×2, and the stride of other convolution layers is set to 1×1.

Baselines. Below, we briefly introduce the baseline methods compared in the experiment.

- *SGD* performs gradient descent on continuously arriving task data, that is, without any constraints on network parameter updates. This approach generally leads to catastrophic forgetting of old tasks after learning new tasks.
- *MTL* uses data from all tasks simultaneously to train the network. It is not a CL setting, and there is no catastrophic forgetting. In addition, MTL can improve the performance of tasks by using knowledge transfer between tasks, and it can be regarded as the upper bound of CL.
- *EWC* [23] uses the diagonal terms of the Fisher information matrix to measure the importance of each parameter for the old tasks, and then uses that importance as the constraint term when the new tasks are learned.
- *MAS* [1], like EWC, considers the parameters' importance to the old tasks as the regular term when the new tasks update the parameters.
- *HAT* [44] uses a hard attention mechanism to generate network parameter masks. The mask lets some parameters remain unchanged while the rest are updated to adapt to the new task.
- *A-GEM* [9] constraints require that the inner product of the gradient direction of the current new tasks on the network weight and the average gradient direction of the old tasks on the current weight be positive.
- *ER* [10] makes a fixed-size memory where only a few raw examples of old tasks are kept. During the learning of new tasks, some samples are selected by reservoir sampling. These selected samples and the samples of new tasks form a new batch to train the model.

Table 6: The hyperparameters for the baselines and our approach. The 'lr' denotes the initial learning rate. We represent PMNIST as 'MNIST', 10-Split CIFAR-100 as 'CIFAR', 5-Datasets as 'FIVE' and Split miniImageNet as 'MIIMG'.

Methods	Hyperparameters
SGD	lr : 0.01 (MNIST, CIFAR), 0.1 (FIVE,MIIMG)
MTL	lr : 0.05 (CIFAR), 0.1 (MNIST, FIVE, MIIMG)
EWC	lr : 0.03 (MNIST, FIVE, MIIMG), 0.05 (CIFAR); regularization coefficient : 1000 (MNIST), 5000 (CIFAR, FIVE, MIIMG)
MAS	lr : 0.001 (MNIST), 0.005 (CIFAR), 0.1 (FIVE, MIIMG); regularization coefficient: 1 (MNIST, MIIMG), 2 (FIVE, MIIMG)
HAT	lr : 0.03 (MIIMG), 0.05 (CIFAR), 0.1 (FIVE); s_{max} : 400 (CIFAR, FIVE, MIIMG); c : 0.75 (CIFAR, FIVE, MIIMG)
A-GEM	lr : 0.05 (CIFAR), 0.1 (MNIST, FIVE, MIIMG); memory size: 1000 (MNIST), 2000 (CIFAR), 3000 (FIVE), 500 (MIIMG)
ER	lr : 0.05 (CIFAR), 0.1 (MNIST, FIVE, MIIMG); memory size: 1000 (MNIST), 2000 (CIFAR), 500 (MIIMG), 3000 (FIVE)
OWM	lr : 0.3 (MNIST), 0.01 (CIFAR)
GPM	lr : 0.01 (MNIST, CIFAR), 0.1 (FIVE, MIIMG); representation size : 100 (FIVE, MIIMG), 125 (CIFAR), 300 (MNIST)
FS-DGPM	lr : 0.01 (MNIST, CIFAR), 0.1 (MIIMG); Memory size: 1000 (MNIST, CIFAR, MIIMG)
GPM+CPR	lr : 0.01 (MNIST, CIFAR), 0.1 (FIVE, MIIMG); regularization coefficient: 0.5 (MNIST, CIFAR, FIVE, MIIMG)
DFGP &	lr : 0.01 (MNIST, CIFAR), 0.1 (FIVE, MIIMG); α: 20 (MNIST, CIFAR, FIVE, MIIMG); λ: 0.1 (MNIST, CIFAR), 0.001 (FIVE), 0.01 (MIIMG); ρ: 0.05 (MNIST, CIFAR, FIVE, MIIMG)

- *OWM* [57] stores the input representation of all samples of the old task as the subspace that the old tasks cover. When new tasks are learned, they are updated in the direction orthogonal to these subspaces.
- *GPM* [43] is similar to OWM, and the new task is also updated along the direction orthogonal to the old tasks across the subspace; the difference is that it stores the basis of the subspace instead of the entire subspace.
- *FS-DGPM [12]* adds a learnable coefficient to the basis in GPM and uses adversarial weight perturbation [55] to improve the flatness of the network. In addition, FS-DGPM also uses an extra memory to store raw samples of old tasks.
- *GPM+CPR*. CPR [6] improves the generalizability of the classifier by maximizing the entropy of the output probability [39] of the classifier to make the prediction more evenly distributed. We added the CPR strategy to the GPM approach as a stronger baseline.

Implementation details. We follow the baseline method's hyperparameter settings in GPM [43] and we list the hyperparameter configuration for all methods in Tab. 6. Some special settings are described below. For the MAS [1] baseline, we search for the regularized strength hyperparameter in $[0.01, 0.1, 1, 5]$. For the GPM, FS-DGPM and our DFGP, we search the projection threshold ϵ_{th}^{l} from $[0.90, 0.91, \ldots, 0.99]$, with an interval of 0.1. For our DFGP, we search for λ for the loss on perturbed data in $[0.001, 0.01, 0.1]$. We set the α parameter of the Beta distribution for the data perturbation to 20 and the weight perturbation radius ρ to 0.05 or 0.1. For each method, we run five random seeds and report the mean and standard deviation.

Training details. We follow the training settings of GPM [43], including training epochs and batch size for each dataset. Specifically, for *PMNIST*, we trained 5 epochs per task and set the batch size to 10. For *CIFAR-100*, we trained

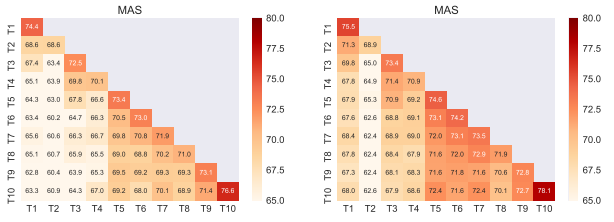

Figure 7: The accuracy (Higher Better) on the CIFAR dataset. (a) MAS; (b) DF-MAS. t-th row represents the accuracy of the network tested on tasks $1-t$ after task t is learned.

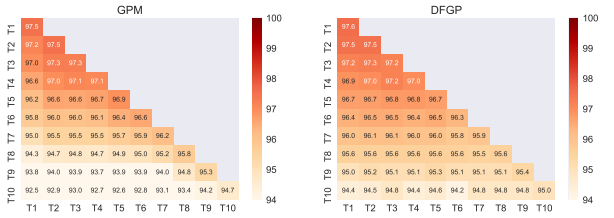

Figure 8: The accuracy (Higher Better) on the PMNIST dataset. (a) GPM; (b) DFGP. t-th row represents the accuracy of the network tested on tasks $1-t$ after task t is learned.

200 epochs per task and set the batch size to 64. For *5-Datasets and MiniImageNet*, we trained 100 epochs per task and set the batch size to 64. All datasets use SGD as the base optimizer. In particular, for CIFAR-100, 5-Datasets and MiniImageNet, when the loss on the validation set is greater than the optimal loss for five consecutive times, we halve the learning rate.

B. Definition and More Experimental Results

B.1. The Definition of Flatness

Below, we explain why the maximum eigenvalue of the Hesse matrix can show how flatness the loss surface is. A flat loss surface is broadly defined as where the loss surface

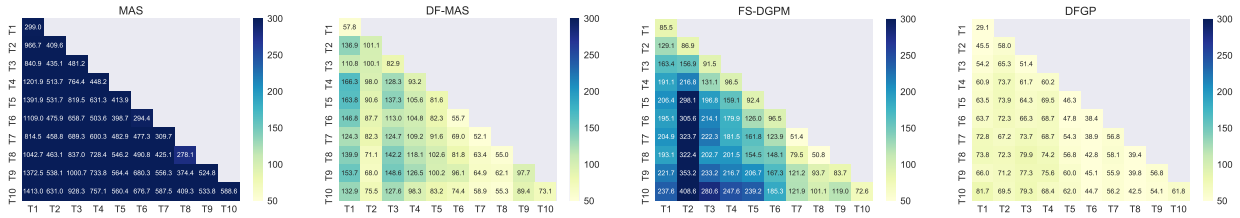

Figure 9: The maximum eigenvalue (Lower Better) on the CIFAR-100 dataset. (a) MAS; (b) DF-MAS; (c) FS-DGPM; (d)DFGP(ours). t-th row represents the maximum eigenvalue of the network tested on tasks $1-t$ after task t is learned.

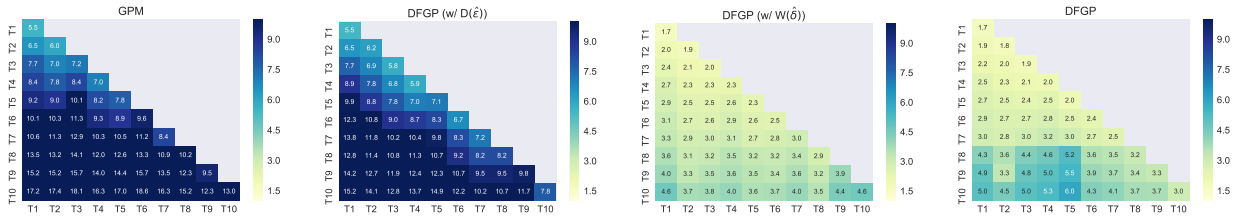

Figure 10: The maximum eigenvalue (Lower Better) on PMINST dataset. (a) GPM; (b) DFGP (w/ D($\hat{\epsilon}$)); (c) DFGP (w/ W($\hat{\delta}$)); (d) DFGP. t-th row represents the accuracy of the network tested on tasks $1-t$ after task t is learned.

does not change drastically with weights within a certain neighborhood [20, 16, 22]. A specific example is given in Fig. 2. That is, if the value F in the equation below is smaller, the loss surface is flatter.

$$F := \max_{\|\boldsymbol{\delta}\|_2 \leq \rho} \mathcal{L}\left(\boldsymbol{W}+\boldsymbol{\delta}, \boldsymbol{X}, \boldsymbol{Y}\right) - \mathcal{L}\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right),$$

where ρ is a hyper-parameter that represents the radius of the neighborhood. According to the second-order Taylor expansion, we have the following:

$$\begin{aligned}\mathcal{L}\left(\boldsymbol{W}+\hat{\boldsymbol{\delta}}, \boldsymbol{X}, \boldsymbol{Y}\right) \approx & \mathcal{L}\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) + \nabla \mathcal{L}\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) \hat{\boldsymbol{\delta}} \\ & + \frac{1}{2} \hat{\boldsymbol{\delta}} \nabla \mathcal{L}^2\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right) \hat{\boldsymbol{\delta}},\end{aligned}$$

where $\nabla \mathcal{L}^2\left(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}\right)$ is the Hessian matrix w.r.t. weight $\boldsymbol{W}$, and is positive semi-definite at a local minimum. $\hat{\boldsymbol{\delta}}$ is the perturbation in the neighborhood of $\boldsymbol{W}$ that causes the worst-case loss. At a local minimum $\boldsymbol{W}^*$, $\nabla \mathcal{L}\left(\boldsymbol{W}^*, \boldsymbol{X}, \boldsymbol{Y}\right) \hat{\boldsymbol{\delta}} = 0$, hence we have

$$F \approx \frac{1}{2} \hat{\boldsymbol{\delta}} \nabla \mathcal{L}^2\left(\boldsymbol{W}^*, \boldsymbol{X}, \boldsymbol{Y}\right) \hat{\boldsymbol{\delta}} \leq \frac{1}{2} \lambda_1^{max} \hat{\boldsymbol{\delta}}^2,$$

where λ_1^{max} is the maximum eigenvalue. In other words, we can use the maximum eigenvalue of the Hessian matrix w.r.t. $\boldsymbol{W}$ to measure the flatness of the loss surface.

B.2. Results on CIFAR-100 Dataset

In this section, we analyze that the proposed Data-augmented Flatness-aware optimization (abbreviated as DF) strategy does improve the plasticity of MAS [1] as well as the flatness of the loss surface. Furthermore, our flatness comparison with FS-DGPM [12] and training time analysis shows that the proposed DFGP is more efficient and achieves a flatter loss surface. Finally, we also analyze the adversarial robustness of each component of DFGP.

Stability analysis. As shown in Tab. 3 of Sec. 4.3, we combined the proposed DF into an orthogonal projection based TRGP [27], a regularized-based MAS [1] method and a memory-based ER [10] method to verify that DF can further improve their performance. In this section, we take the MAS method on the CIFAR-100 dataset as an example to further analyze whether DF-MAS alleviates the catastrophic forgetting problem of MAS by improving the flatness of the loss surface. As shown in Fig. 7, after learning task $T10$, the test accuracy of the network in tasks $T1$, $T5$ and $T7$ under the DF-MAS method is 68.0%, 72.4% and 72.4% respectively. However, in the corresponding tasks, MAS can only obtain test accuracy of 63.3%, 69.2% and 70.1% respectively. The DF-MAS improved by almost two percentage points on the old tasks compared to the MAS, indicating that the DF-MAS effectively mitigated catastrophic forgetting.

Flatness visualization. According to Sec. B.1, below we visualize the flatness of the loss surface under the MAS method and the DF-MAS method. As shown in Fig. 9(a) and (b), we can observe that the maximum eigenvalue in MAS after learning task $T10$ is $5.97-10.63$ times that of DF-MAS. This indicates that the loss surface of the MAS method is sharper than that of the DF-MAS method. We also compare the flatness of our method with FS-DGPM in Fig. 9(c) and (d). We can observe that the maximum eigenvalues of FS-DGPM are almost all smaller than GPM (Fig. 6(c)), however, there is still some gap with our DFGP, that is, the loss surface is not as flat as ours. This is because we perform flatness optimization at both the data and weight levels, while FS-DGPM merely considers the weight level. Also, the manner in which flatness-aware optimization is solved may also be a contributing factor to the difference.

Training time. We provide a time-to-convergence comparison of GPM, FS-DGPM, and our DFGP on CIFAR-100. The experimental results are shown in Tab. 7, GPM is the most ef-

ficient, but because it does not consider the optimization goal of flatness, it limits its performance (Tab. 1). FS-DGPM is the least efficient because it iteratively needs to perform iterative adversarial weight perturbations [55] on new task data and old task data in memory and needs to update the scale of the base. Finally, our method achieves acceptable speed and state-of-the-art performance thanks to the approximate solution of our data perturbations and weight perturbations. Of course, we could speed up DFGP by performing flatness-aware optimization periodically instead of at every step [28].

Table 7: Analysis of training time on CIFAR-100.

Methods	GPM	FS-DGPM	DFGP(ours)
Time Cost (min)	8.94	152.65	23.25

A discussion of radius ρ. ρ is a hyperparameter representing the radius of the neighborhood. As shown in Tab. 8, when ρ is very small (e.g. 0.01), indicating a small radius of data and weight perturbation, the effect of flatness optimization is minimal. However, it still outperforms GPM (i.e., 72.31), and performance may improve if ρ is increased. ρ at 0.05 is a solid default, which is consistent with previous work [16].

Table 8: Analysis of hyperparameter ρ on CIFAR-100.

ρ	0.01	0.025	0.05	0.075	0.1
ACC(%)	73.00±0.77	73.61±0.31	74.59±0.33	74.14±0.34	74.04±0.34

B.3. Results on PMNIST Dataset

In this section, we analyze the flatness of GPM, DFGP(w/ D($\hat{\epsilon}$)), DFGP(w/ W($\hat{\delta}$)), and DFGP on the PMNIST dataset.

Stability-Plasticity analysis. As shown in Fig. 8, compared with GPM, DFGP has more advantages in the performance of new tasks and significantly alleviated the degree of forgetting of old tasks.

Flatness visualization. As shown in Fig. 10(a) and (d), the maximum eigenvalue in DFGP is smaller than GPM. Of course, we also observed that the maximum eigenvalue on the PMNIST dataset is much smaller than that on other datasets. This is because of the high similarity between continuously arriving tasks in the PMNIST dataset; they have the same distribution of labels, just different permutations of input pixels. In addition, the simplicity of the network architecture in PMNIST compared to other datasets is also a reason. As shown in Fig. 10(b) and (c), we also analyze the flatness improvement at the two levels of data and weights in DFGP. Compared with GPM in Fig. 10(a), our two strategies have smaller maximum eigenvalues, i.e., loss surface is flatter.

B.4. Results on MiniImageNet Dataset

In this section, we analyze the performance and flatness of GPM and DFGP on the MiniImageNet dataset.

Stability-Plasticity analysis. As shown in Fig. 11, after learning the 20-th task, the accuracy of the network on old tasks $T5$, $T10$, and $T15$ in the GPM method is 57%, 57%, and 67%, respectively. However, DFGP can achieve 65%, 65%, 74%. In addition, as shown in Fig. 13, DFGP also basically outperforms GPM on new tasks that arrive continuously. However, there is still a gap between our proposed CL method and the multi-task joint training (MTL method), and we need to further explore more effective CL methods to narrow this gap in the future.

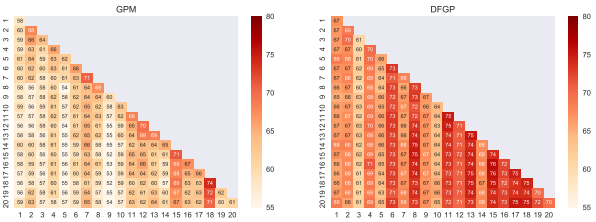

Figure 11: The accuracy (Higher Better) on the MiniImageNet dataset. (a) GPM; (b) DFGP. t-th row represents the accuracy of the network tested on tasks $1-t$ after task t is learned.

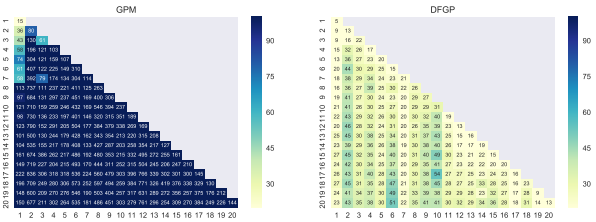

Figure 12: The maximum eigenvalue (Lower Better) on the MiniImageNet dataset. (a) GPM; (b) DFGP. t-th row represents the maximum eigenvalue of the network tested on tasks $1-t$ after task t is learned.

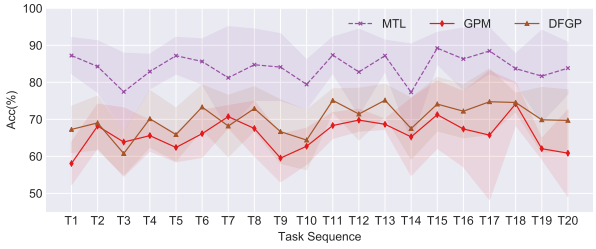

Figure 13: New tasks' accuracy of MTL, GPM, and DFGP on MiniImageNet dataset.

Flatness visualization. As shown in Fig. 12, we can see that on the more challenging MiniImageNet dataset (as shown in Tab. 5, which has more tasks than other datasets) and the more complex ResNet18 network, the loss surfaces in GPM tend to become sharper. For example, in GPM, the flatness of task $T1$ and task $T2$ was 15 and 80 respectively, when they were just trained, while after task $T20$ was learned, the sharpness for these two tasks increased 10.0 times and 8.4 times, that is, to 150 and 677, respectively. However, in DFGP, the two tasks only increased by 4.6 and 3.3 times, respectively. This also suggests that our approach is better at avoiding the flatness of forgetting.