Preserving Linear Separability in Continual Learning by Backward Feature Projection

Qiao Gu
University of Toronto
qgu@cs.toronto.edu

Dongsub Shim
LG AI Research
dongsub.shim@lgresearch.ai

Florian Shkurti
University of Toronto
florian@cs.toronto.edu

Abstract

Catastrophic forgetting has been a major challenge in continual learning, where the model needs to learn new tasks with limited or no access to data from previously seen tasks. To tackle this challenge, methods based on knowledge distillation in feature space have been proposed and shown to reduce forgetting [16, 18, 26]. However, most feature distillation methods directly constrain the new features to match the old ones, overlooking the need for plasticity. To achieve a better stability-plasticity trade-off, we propose Backward Feature Projection (BFP), a method for continual learning that allows the new features to change up to a learnable linear transformation of the old features. BFP preserves the linear separability of the old classes while allowing the emergence of new feature directions to accommodate new classes. BFP can be integrated with existing experience replay methods and boost performance by a significant margin. We also demonstrate that BFP helps learn a better representation space, in which linear separability is well preserved during continual learning and linear probing achieves high classification accuracy.

1. Introduction

Despite their many successes, deep neural networks remain prone to catastrophic forgetting [38], whereby a model's performance on old tasks degrades significantly while it is learning to solve new tasks. Catastrophic forgetting has become a major challenge for continual learning (CL) scenarios, where the model is trained on a sequence of tasks, with limited or no access to old training data. The ability to learn continually without forgetting is crucial to many real-world applications, such as computer vision [37, 49], intelligent robotics [31], and natural language processing [6, 24]. In these settings, an agent learns from a stream of new data or tasks, but training on the old data is restricted due to limitations in storage, scaling of training time, or even concerns about privacy.

The continual learning problem has received significant attention and multiple solution themes have emerged. Experience replay methods [8, 34], for example, store a limited number of (or generate) old training examples and use them together with new data in continual learning. Parameter regularization methods [30, 53] restrict the change of important network parameters. Knowledge distillation methods [16, 18, 32] regularize the intermediate output of the CL model to preserve the knowledge from old tasks. Architectural methods [35, 43, 51] adopt expansion and isolation techniques with neural networks to prevent forgetting. All these methods strive to balance learning new knowledge (plasticity) and retaining old knowledge (stability).

We present a continual learning algorithm, focusing on knowledge distillation (KD) in feature space. In the continual learning context, KD treats the continual learning model as the student and its old checkpoint as the teacher and regularizes the network intermediate outputs to reduce forgetting [4, 8, 11, 16, 18, 26, 32]. Although recent CL meth-

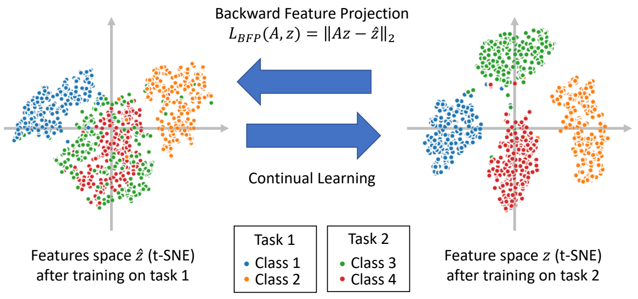
Backward Feature Projection
$L_{BFP}(A, z) = \|Az - \hat{z}\|_2$
Continual Learning
Features space $\hat{z}$ (t-SNE) after training on task 1
Task 1
● Class 1
● Class 2
Task 2
● Class 3
● Class 4
Feature space z (t-SNE) after training on task 2

Figure 1. Feature distribution before and after training on a task in a class incremental learning experiment on MNIST, visualized by t-SNE. **Left**: before training on task 2, seen classes (1,2) are learned to be separable along the horizontal axis for classification, while unseen classes (3, 4) are not separable. **Right**: after training on task 2, the new vertical axis is learned to separate new classes (3,4). Based on this observation, we propose the Backward Feature Projection loss L_{BFP}, which allows new feature dimensions to emerge to separate new classes in feature space and also preserves the linear separability of old classes to reduce catastrophic forgetting.

ods based on KD have been effective at reducing forgetting, they typically adopt the L_2 distance for distillation, forcing the learned features to be close to their exact old values. This is too restrictive and results in CL models that are more rigid in retaining old knowledge (stronger stability), but less flexible in adapting to new tasks (weaker plasticity). Our method has a better tradeoff of stability and plasticity.

In this paper, we pay attention to the feature space in CL and study its evolution. We show that a small number of principal directions explain most of the variance in feature space and only these directions are important for classification. A large number of directions in the feature space have little variance and remain unused. When the model is trained on new tasks, new features need to be learned along those unused directions to accommodate new classes, as illustrated in Figure 1. Without handling forgetting, the old principal directions, along which the old classes are linearly separable, will be forgotten. Our results indicate that such forgetting of learned principal directions in the feature space is an important reason for catastrophic forgetting.

Based on this insight, as shown in Figure 1, we propose a Backward Feature Projection (BFP) loss, an effective feature distillation loss that enforces feature consistency up to a learnable linear transformation, not imposing exact equality of features. This transformation aims to preserve the linear separability of features backward in time. We show that this linear projection is important because it can rotate, reflect, and scale features, while maintaining the linear separability of the previously learned classes in the new feature space. Projecting backward allows the features to change and new decision boundaries to be learned along the unused feature directions to classify new classes. BFP can be integrated into existing CL methods in a straightforward way and experiments show that this simple change boosts the performance over baselines by a large margin.

Our experiments show that the proposed BFP regularization loss can improve the baseline methods by up to 6%-8% on the challenging Split-CIFAR10 and Split-CIFAR100 datasets, achieving state-of-the-art class-incremental learning accuracy. More importantly, the linear probing experiments show that BFP results in a better feature space where different classes are more separable. See Figure 1 for an illustrative example. Our contributions are as follows:

- We provide an analysis of feature space evolution during continual learning, distinguishing the important feature components from unimportant ones.
- We propose the Backward Feature Projection (BFP) loss, which preserves the linear separability of old classes while allowing plasticity during continual learning, i.e. features are allowed to change.
- When combined with simple experience replay baselines, BFP helps learn better feature space and achieves state-of-the-art performance on challenging datasets.

2. Related Work

2.1. Experience Replay Methods

Experience replay or rehearsal methods use a small memory buffer to keep the training data of old tasks. When the model is training on the new task, old training examples are extracted and trained together with new ones. Recent replay-based CL approaches mainly differ in three components, namely which examples to store, how examples are replayed, and how to update the network using old examples. Recent work has focused on evolving the three components mentioned above. ICaRL [41] chooses examples into the memory such that the mean in the feature space of the memory buffer matches that of training data. MIR [2] prioritizes the replay of the examples that are mostly interfered with by a virtual update on the network parameters. DER/DER++ [8] augments the cross entropy loss with a logit distillation loss when the memory data is replayed. GEM [34] and A-GEM [13] develop optimization constraints when trained on new tasks using the old data from memory. Some other work [33, 45, 48] also learn to generate images for replay during CL, but the continual training of the generation network adds some additional challenges. Although the idea is straightforward, experience replay methods often achieve better performance than other types of methods, which marks the importance of storing the old data.

2.2. Parameter Regularization Methods

Parameter regularization methods study the effect of neural network weight changes on task losses and limit the movement of important ones, which would otherwise cause forgetting on old tasks. This line of work typically does not rely on a replay buffer for old task data. One of the pioneering works along this line is EWC [27], which proposed to use the empirical Fisher Information Matrix to estimate the importance of the weight and regularize the weight change in continual learning. SI [53] uses the estimated path integral in the optimization process as the regularization weight for network parameters. MAS [1] improves this idea by adopting the gradient magnitude as a sensitivity measure. RWalk [12] combines Fisher information matrix and online path integral to approximate the parameter importance and also keeps a memory to improve results.

2.3. Knowledge Distillation Methods

Originally designed to transfer the learned knowledge of a larger network (teacher) to a smaller one (student), knowledge distillation methods have been adapted to reduce activation and feature drift in continual learning. Different from parameter regularization methods that directly regularize the network weights, KD methods regularize the network intermediate outputs. Li *et al*. [32] proposed an approach called Learning without Forgetting (LwF), regularizing the output logits between the online learned model and the old checkpoint. DER/DER++ [8] combines this logit regularization with experience replay and further improves the performance. Later Jung *et al.* [25] proposed to do knowledge distillation on the feature maps from the penultimate layer and freeze the final classification layer. Pooled Output Distillation (PODNet) [18] extended the knowledge distillation method to intermediate feature maps, and studied how different ways of feature map pooling affect continual learning performance. They proposed to pool the feature maps along the height and weight dimensions respectively to achieve a good stability-plasticity trade-off. Recent work [16,26] also used gradient information (e.g. Grad-CAM) as weighting terms in the feature distillation loss, such that feature maps that are important to old tasks will change less during continual learning.

Different from existing KD methods, we use a learnable linear layer to project new features to the old ones. This idea was explored in [20,21], but their work only integrated it in a contrastive learning framework and used a nonlinear mapping function. However, in this work, we use a learnable *linear* transformation and formulate it as a simple knowledge distillation loss in the feature space. We demonstrate that our method promotes linear separability in feature space during continual learning. We also show the value of BFP in the supervised CL setting with experience replay, and no augmentation or contrastive learning is needed.

3. Method

3.1. Setting and Notation

In a typical continual learning setting, a model f is sequentially trained on a set of tasks $\mathcal{T} = \{1, 2, 3, \cdots, T\}$. Within each task t, input x and the corresponding ground truth output y are drawn i.i.d. from the task data distribution $\mathcal{D}_t = (X_t, Y_t)$ and used to train the model. Here X_t and Y_t denote the set of inputs and outputs from task t. To illustrate our method, the CL model is decomposed into two parts $f_\theta(x) = g_\phi(h_\psi(x)) = g_\phi \circ h_\psi(x)$ with $\theta = \{\phi, \psi\}$, where h, parameterized by ψ, is a non-linear feature extractor, mapping input image x to a low-dimensional feature $z \in \mathbb{R}^d$. The classification head g, parameterized by ϕ, is a linear layer that maps the latent feature z to classification logits $o \in \mathbb{R}^c$, where c is the total number of classes. In this paper, we mainly consider the class incremental learning (Class-IL) setting and the task incremental learning (Task-IL) setting, and the proposed method works on both settings. In these settings, $\mathcal{D}_t$ contains training data from a set of classes $\mathcal{C}_t$, where $\mathcal{C}_t$ are disjoint for different task t. In Task-IL, task identifiers t for each input are available during evaluation time, and thus the model can focus

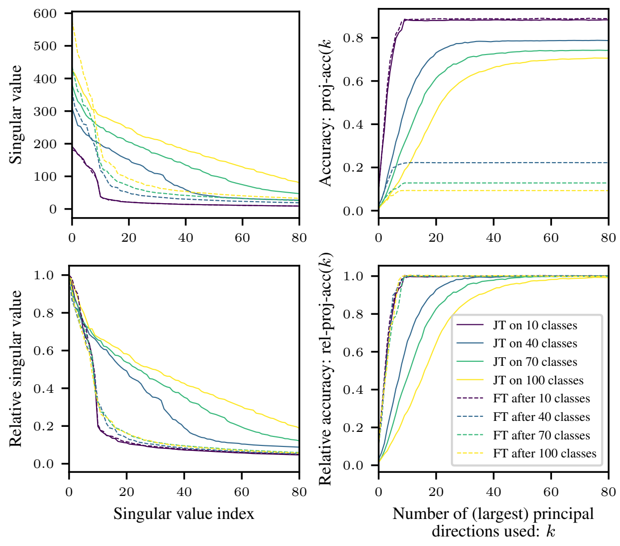

Figure 2. Feature distribution and contribution to classification during continual learning on Split-CIFAR100 with 10 classes per task. **Left**: feature variance (singular values) along each principal direction; **right**: classification accuracies proj-acc(k) using projected features. **Upper** plots show the absolute singular values and accuracies, and **lower** ones show their relative values, normalized by the maximum on each curve. Finetuning (FT) is a naive CL baseline that does not handle forgetting at all and gives a lower-bound performance. Joint Training (JT) is an oracle CL method that is jointly trained on all classes seen so far and shows the upper bound. Contrasting JT with FT reveals the ideal properties for good CL methods. As the model is continually trained on more classes, more feature dimensions are learned and needed for classification. However, compared to the full feature dimension (512), only a small subspace (around 10 principal directions for 10 classes and 80 for 100 classes) is crucial for CL performance, as the relative accuracies quickly saturate with the number of principal directions used.

the decision boundaries within each task. On the contrary, Class-IL requires making a decision among all classes during inference time and thus is more challenging. We further denote the model after training on task j as $f^j = g^j \circ h^j$, and the feature extracted by h^j from a datapoint in task i as $z_i^j = h^j(x),\ x \in \mathcal{D}_i$. The set of all z_i^j forms a feature matrix $Z_i^j = h^j(\mathcal{D}_i) \in \mathbb{R}^{d \times n}$, and n is the number of datapoints in $\mathcal{D}_i$. And similarly, the set of features extracted from D_1 to D_i using h_j is denoted by $Z_{1:i}^j = h_j(\mathcal{D}_{1:i})$.

3.2. Analyzing Feature Forgetting in CL

Motivated by recent work showing that representation drifts in the feature space have been a major cause for catastrophic forgetting [10, 19, 52], we study the evolution of feature space in CL and answer two key questions: (1) how many dimensions (principal directions) in the learned feature space are occupied by the data? And (2) how many of them are used for classification? We answer the first question by conducting principal component analysis (PCA) [39] on the feature matrix $Z^t_{1:t}$, which contains feature extracted by h^t from all data seen so far $\mathcal{D}_{1:t}$. Suppose its singular vector decomposition gives $Z^t_{1:t} = USV^T$, and then principal directions are the left singular vectors $U = [u_1, u_2, \cdots, u_d]$, where $u_l \in \mathbb{R}^d$ are sorted by the corresponding singular values s_l. PCA gives the data distribution of seen tasks in the feature space and thus answers the first question. The second question is answered by evaluating the features projected onto a subspace spanned by the first k principal directions $U_{1:k}$. Specifically, we define the classification accuracy of the projected features as

$$\text{proj-acc}(k) = \text{acc}\left(y, g^t(U_{1:k}U^T_{1:k}z)\right) \tag{1}$$

where k is the number of largest principal components used and proj-acc is computed over the testing set of task t. With a larger k, more information is retained in the projected feature $U_{1:k}U^T_{1:k}z$ and used for classification. The changes of proj-acc with the increase of k reflect the importance of each principal direction being added.

In Figure 2, we plot s_k and proj-acc(k) versus k when a model has been trained on a certain number of classes during CL. We compare two simple CL methods: finetuning (FT) where the model is continually trained on the online data stream without any means to reduce catastrophic forgetting, and joint training (JT) where all the training data seen so far is used to train the network. Typically, FT serves as a naive lower bound for CL performance and JT an oracle upper bound. Contrasting FT and JT reveals the difference in feature space obtained from the worst and the ideal CL methods.

We can see from Figure 2, for JT, as the network is trained on more classes, the learned features span a larger subspace and the classifier needs more principal directions to achieve good classification accuracies (high relative proj-acc). This shows that during continual learning, more feature directions are needed to make new classes linearly separable in the feature space. However, for the naive FT baseline, the number of principal directions with large variance does not increase with the number of seen classes. This indicates feature forgetting: a poor CL method only focuses on the feature directions that are important for the current task. The feature directions for old tasks are suppressed to low variance and thus forgotten. On the other hand, compared to the full feature dimension $d = 512$, JT accuracies still saturate with a relatively small $k = 80$, which is roughly the number of classes seen so far. Other feature directions that have low variance are not used for classification, and such "unused" feature directions could leave room for future tasks in CL.

Based on this insight, we argue for the benefit of preserving important feature directions for old tasks while allowing new ones to emerge for new tasks during continual learning. Therefore, we propose to learn a *linear* transformation that projects the new feature space back to the old one and in the following section, we show it can achieve both goals.

3.3. Backward Feature Projection

We denote the feature extractor that is being trained on the current task t as h, which may not have converged, and the converged model checkpoint at the end of the last task as $h' = h^{t-1}$. Given an input example x, the extracted features are denoted as $z = h(x)$ and $z' = h'(x)$. To preserve information in feature space such that the new feature z should contain at least the information as that in z', we can learn a projection function p that satisfies $z' = p(z)$ [20, 21].

In this work, we propose that a *linear* transformation matrix A can well preserve linear separability and suffice to reduce forgetting. Formally, we propose the Backward Feature Projection (BFP) loss in continual learning. Given a training example x,

$$\begin{aligned} L_{\text{BFP}}(x; \psi, A) =& \|Az - z'\|_2 \quad (2) \\ =& \|Ah_\psi(x) - h'(x)\|_2, \quad (3) \end{aligned}$$

where we omit the bias term by adding a fixed entry of 1 in the feature vector z. Here we only optimize h_ψ and A, while we freeze h' and thus omit its parameters.

In the following, we show that the BFP loss can preserve the linear separability of old classes while allowing new classes to be classified along the unused directions in the old feature space. Consider the extracted features from any two examples from the old classes $z'_i = h'(x_i),\ x_i \in C_1$ and $z'_j = h'(x_j),\ x_j \in C_2$, where $C_1, C_2 \in \mathcal{C}_{t-1}$. If they are learned to be linear separable at the end of task $t-1$, then there exists a vector w and a threshold b, such that

$$w^T z'_i > b > w^T z'_j,\ \forall i \in C_1,\ \forall j \in C_2. \tag{4}$$

Then if the BFP loss in Equation 3 is well optimized, i.e. $z' \approx Az$ with a linear transformation A. Then for the features extracted from h,

$$\begin{aligned} w^T Az_i >& b > w^T Az_j,\ \forall i \in C_1,\ \forall j \in C_2 \quad (5) \\ \Rightarrow (A^T w)^T z_i >& b > (A^T w)^T z_j,\ \forall i \in C_1,\ \forall j \in C_2. \quad (6) \end{aligned}$$

Therefore the feature vectors z from the old classes C_1, C_2 remain linearly separable along the direction $A^T w$ in the new feature space. The linear classifier g is trained to find this decision boundary during CL with experience replay.

To classify new classes, the network needs to map them to linearly separable regions in the feature space. The linear transformation in BFP achieves it by arranging the new classes along the "unused" feature directions that have low variance and thus are not occupied by the old tasks. Consider that the features extracted from future task $\mathcal{D}_t$ using the old model h' are probably not separable and mixed together. This is natural as h' has not been trained on it. As we can see from Section 3.2 and Figure 2, there exists many principal directions with low variance, along which features from different classes are not separable, Ideally, a CL model should take up these "unused" feature directions to learn features that are needed to classify new classes. Without loss of generality, suppose before the model is trained on a new task t, the feature extracted from the new task $z' = h'(x)$, $x \in X_t$, are all mapped to zero along an "unused" feature direction v, i.e. $v^T z' = 0$. Then after learning on task t, the feature $z = h(x)$ from new classes $C_3, C_4 \in \mathcal{C}_t$ can be learned to be separable along that feature direction v,

$$v^T z_i > v^T z_j,\ \forall i \in C_3,\ \forall j \in C_4. \tag{7}$$

In this case, A can be learned such that $v \notin \text{Col}(A)$ and thus $v^T(Az) = 0$ while $v^T z \neq 0$ (satisfying Equation 7). In this way, the BFP loss in Equation 3 allows the new class to be separable along v and still can be minimized. Note that during the actual continual learning with BFP, neither w, v nor the dimensionality of them is defined or needed. They are learned implicitly in the matrix A through gradient descent optimization. w and v can be extracted and analyzed by PCA decomposition, but it is not required for training.

3.4. Loss functions

We integrate the proposed backward feature projection method into an experience replay framework [8], where we keep a buffer M storing training examples from old tasks. We focus on experience replay methods because they are simple and outperform other types of CL methods by a large margin according to a recent survey [36]. We keep the model checkpoint at the end of the last task f^{t-1} together with the online trained model f. During continual learning, the online model f is trained on a batch from the online data stream of the current task $\mathcal{D}_t$ using cross-entropy loss.

$$L_{\text{ce}}(\mathcal{D}_t;\theta) = \sum_{x,y\in\mathcal{D}_t} \text{cross-entropy}(y, f_\theta(x)) \tag{8}$$

Meanwhile, we sample another batch from M for experience replay. Following [8], a cross-entropy loss and a logit distillation loss are applied on the replayed data

$$L_{\text{rep-ce}}(M;\theta) = \sum_{x,y\in M} \text{cross-entropy}(y, f_\theta(x)), \tag{9}$$

$$L_{\text{rep-logit}}(M;\theta) = \sum_{x,y\in M} \|f_\theta(x) - f^{t-1}(x)\|_2^2. \tag{10}$$

And we apply our backward feature projection loss on both the online data stream $\mathcal{D}_t$ and the replayed examples M

$$L_{\text{BFP}}(\mathcal{D}_t, M;\psi, A) = \sum_{x,y\in\mathcal{D}_t,M} \|Ah_\psi(x) - h^{t-1}(x)\|_2. \tag{11}$$

The total loss function used in continual learning is the weighted sum of the losses above.

$$\begin{aligned} L(\mathcal{D}_t, M;\theta, A) = L_{\text{ce}}(\mathcal{D}_t;\theta) + \alpha L_{\text{rep-ce}}(M;\theta) \\ +\beta L_{\text{rep-logit}}(M;\theta) + \gamma L_{\text{BFP}}(\mathcal{D}_t, M;\psi, A) \end{aligned} \tag{12}$$

During training on task t, both the linear transformation A and the model f_θ are optimized, and the old model checkpoint f^{t-1} remains fixed. In our experiments, the matrix A is randomly initialized at the beginning of each task. The pseudo-code of the proposed algorithm can be found in the Appendix.

4. Experiments

4.1. Experimental Setting

Continual Learning Settings. We follow [8, 14, 50] and test all methods using both the Class-IL and Task-IL settings in our CL experiments. Both Class-IL and Task-IL split the dataset into a sequence of tasks, each containing a disjoint set of classes, while task identifiers are available during testing under Task-IL. Task-IL thus has extra advantages during inference (e.g. select proper prediction head) and becomes an easier CL scenario. Our work is designed for Class-IL and its Task-IL performance is obtained by only considering the logits within the ground truth task.

Datasets. We evaluate baselines and our methods on the following datasets using varying buffer sizes: **Split CIFAR-10** divides the original CIFAR-10 [29] dataset into 5 tasks, with each task composed of 2 classes. Each class includes 5000 training and 1000 testing images of shape 32×32. **Split CIFAR-100** divides CIFAR-100 [29] into 10 tasks, with 10 classes per task. Each class has 500 training and 100 testing images of shape 32×32. **Split TinyImageNet** splits TinyImageNet [47] into 10 tasks, with 20 classes per task. Each class contains 500 training images, 50 validation images, and 50 testing images. These datasets are challenging and state-of-the-art continual learning methods still fall far behind the Joint Training (JT) baseline, especially in the Class-IL setting as shown in Table 1.

Metrics. Following [3, 7, 8, 36], we report the performance of each compared method using Final Average Accuracy (FAA). Suppose a_i^t is the testing classification accuracy on the i^{th} task when the training finishes on task t, FAA is the accuracy of the final model averaged all tasks:

$$FAA = \frac{1}{T}\sum_{i=1}^{T} a_i^T. \tag{13}$$

We also report the Final Forgetting (FF), which reflects the accuracy drop between the peak performance on one task and its final performance:

$$FF = \frac{1}{T-1}\sum_{i=1}^{T-1} \max_{j\in 1,\cdots,T-1} (a_i^j - a_i^T). \tag{14}$$

Setting	Method	S-CIFAR10		S-CIFAR100		S-TinyImageNet
	Buffer Size	200	500	500	2000	4000
Class-IL	Joint Training (JT)	91.27±0.57		70.68±0.57		59.61±0.25
	Finetuning (FT)	36.20±2.02		9.36±0.07		8.11±0.08
	iCaRL [41]	63.58±1.22	62.62±2.07	46.66±0.23	52.60±0.38	31.47±0.46
	FDR [5]	31.24±2.61	28.72±2.86	22.64±0.56	34.84±1.03	26.52±0.41
	LUCIR [23]	58.53±3.03	70.37±0.97	35.14±0.57	48.95±1.21	29.79±0.70
	BiC [50]	59.53±1.77	75.41±1.14	35.96±1.38	45.44±0.96	15.98±1.01
	ER-ACE [10]	63.54±0.42	71.17±1.38	38.86±0.72	50.20±0.39	37.72±0.16
	ER [42]	58.07±2.92	68.04±2.18	20.34±0.96	37.44±1.48	23.29±0.54
	ER w/ BFP (Ours)	63.27±1.09 (+5.21)	71.51±1.58 (+3.47)	22.54±1.10 (+2.20)	38.92±1.94 (+1.48)	26.33±0.68 (+3.04)
	DER++ [8]	65.41±1.60	72.65±0.33	38.88±0.91	52.74±0.79	41.24±0.64
	DER++ w/ BFP (Ours)	**72.21±0.22** (+6.80)	**76.02±0.79** (+3.37)	**47.45±1.30** (+8.56)	**57.91±0.66** (+5.17)	**43.40±0.41** (+2.16)
Task-IL	Joint Training (JT)	98.19±0.16		91.40±0.43		82.21±0.33
	Finetuning (FT)	65.01±5.12		35.54±2.63		18.46±1.26
	iCaRL [41]	90.27±1.21	90.05±1.46	84.45±0.48	86.24±0.47	66.06±0.75
	FDR [5]	91.42±1.03	93.40±0.31	74.66±0.16	82.15±0.26	66.79±0.66
	LUCIR [23]	94.30±0.79	94.99±0.14	85.13±0.20	87.50±0.44	70.09±0.40
	BiC [50]	95.41±0.73	**96.45±0.34**	**85.16±0.36**	87.03±0.30	68.44±4.40
	ER-ACE [10]	92.12±0.62	94.09±0.27	77.00±0.80	83.30±0.54	68.91±0.38
	ER [42]	92.06±0.65	93.60±0.76	72.42±1.74	81.34±1.06	64.96±0.45
	ER w/ BFP (Ours)	95.50±0.41 (+3.44)	96.11±0.27 (+2.51)	79.79±1.67 (+7.38)	84.16±1.18 (+2.81)	71.43±0.58 (+6.47)
	DER++ [8]	91.71±0.83	93.76±0.27	76.96±0.24	83.59±0.41	71.14±0.53
	DER++ w/ BFP (Ours)	**95.95±0.22** (+4.23)	96.29±0.26 (+2.53)	83.64±0.64 (+6.68)	**87.20±0.32** (+3.61)	**73.07±0.28** (+1.94)

Table 1. Final Average Accuracies (FAA, in %) in Class-IL and Task-IL setting of baselines and our methods on various datasets and buffer sizes. The green numbers in parentheses show the absolute improvements brought by BFP over the corresponding ER or DER++ baselines. The proposed BFP method can boost the Class-IL performance by up to 6-8% in some challenge settings with small buffer sizes. DER++ w/ BFP (Ours) outperforms all baselines in terms of Class-IL accuracies and has Task-IL accuracies that are better or close to the top-performing methods. Mean and standard deviation are computed over 5 runs with different seeds. Joint Training (JT) shows the upper bound performance, where the model is trained on data from all tasks and Finetune (FT) is the lower bound, where the model is trained sequentially without handling forgetting.

Lower FF means less forgetting and better CL performance.

Training details. We use ResNet-18 [22] as the network backbone, and instead of the simple reservoir buffer used in [8], we use class-balanced reservoir sampling [9] for pushing examples into the buffer. All the baselines we compare are updated with this change. We use an SGD optimizer to optimize the model f_θ and another SGD+Momentum optimizer with a learning rate 0.1 for the projection matrix A. The optimizers and the matrix A are re-initialized at the beginning of each task. The network is trained for 50 epochs per task for Split CIFAR-10 and Split CIFAR-100 and 100 epochs per task for Split Tiny-ImageNet. The learning rate is divided by 10 after a certain number of epochs within each task ($[35, 45]$ for Split CIFAR-100 and $[35, 60, 75]$ for Split TinyImageNet). In this work, we focus on this offline CL setting where each task is trained for multiple epochs. Although we are also interested in online CL, multi-epoch training helps disentangle underfitting and catastrophic forgetting [3, 8]. BFP introduces only one extra hyperparameter γ, which is set to 1 for all experiments. We found that $\gamma = 1$ works well for all datasets and buffer sizes and did not perform hyperparameter searches for individual experiment settings. Hyperparameters α and β used in Equation 12 are adopted from [8]. Most baselines adopt different hyperparameters for different settings, for which we adopt the hyperparameters that have been optimized by grid search by [8] and [7] for a fair comparison. The details can be found in the Appendix.

4.2. Baselines

First, we evaluate the performance of **Joint Training (JT)** and **Finetuning (FT)** baselines on each dataset. JT trains the network on all training data, does not have the forgetting problem, and thus indicates the upper-bound performance of CL methods. On the contrary, FT simply performs SGD using the current task data without handling forgetting at all and indicates the lower-bound performance.

As a feature distillation method, our method can be combined with most continual learning methods. In our evaluation, we test our method by combing it with two popular experience replay methods, ER [42] and DER++ [8]. **ER** uses a memory buffer to store the training examples from past tasks and interleaves them with the current task data for training. In addition to this, **DER++** records the output logits of the examples in the memory and performs logit distillation when doing experience replay. We combine the proposed BFP loss with ER and DER++ and denote them as **ER w/ BFP** and **DER++ w/ BFP** respectively.

We also compare the proposed method with some other state-of-the-art CL baselines as listed in Table 1. **Incremental Classifier and Presentation Learning (iCaRL)** [41] performs classification using the nearest mean-of-exemplars, where the exemplars selected by herding algorithm in the feature space. **Functional Distance Regularization (FDR)** [5] regularize the output of the network to its past value. Different from DER/DER++, FDR applies the regularization on the output classification probability. **Learning a Unified Classifier Incrementally via Rebalancing (LUCIR)** [23] augments experience replay with multiple modifications to preserve old knowledge and enforce separation class separation in continual learning. **Bias Correction (BiC)** [50] augments the experience replay by learning a separate layer to correct the bias in the output logits. **ER with Asymmetric Cross-Etronpy (ER-ACE)** [10] proposes to reduce representation drift by using separate cross-entropy loss for online and replayed training data.

4.3. Results

The Final Average Accuracies in the Class-IL and Task-IL settings are reported in Table 1. The corresponding table for Final Forgetting can be found in the Appendix. We test the methods on three datasets with various sizes of memory buffers. Experiments are averaged over 5 runs with different seeds and mean and standard deviation are reported. First, we observe that there is a still big gap between the current best CL methods and the JT oracle baselines on all datasets, especially in the Class-IL setting, which indicates that CL is still an unsolved and challenging problem. Comparing DER++ and DER++ w/ BFP, we can see that BFP boosts the Class-IL accuracies by a significant margin, especially with a small buffer size (6.8% on S-CIFAR10 with buffer size 200 and 8.5% on S-CIFAR100 with buffer size 500). DER++ w/ BFP thus outperforms all baseline methods in the Class-IL setting, which are very challenging as the final model needs to distinguish testing examples from all seen classes. Previous CL methods struggle to have satisfactory performance in this setting. Under the Task-IL setting that is easier because task identifiers are known during evaluation time, our model also helps achieve much higher accuracies over the base ER or DER++ method. And among all the CL methods compared, the proposed method also achieves the best or close-to-best accuracies under the Task-IL setting.

4.4. Linear Probing

Some latest work on continual learning studied catastrophic forgetting in the final feature space $h^T(x)$ [15, 55]. They show that although the accuracy using the continually trained classifier g^T degrades heavily due to catastrophic forgetting, the learned knowledge in h^T is well maintained. This is shown by fitting a linear classifier g^* on top of the frozen feature extractor at the end of continual learning h^T. Such *linear probing accuracies* obtained by $g^* \circ h^T$ can be much higher than $g^T \circ h^T$. Therefore recent work argues that catastrophic forgetting mainly happens at the last linear classification layer and the linear probing accuracies can be used as a quality measure for the representation learned from continual learning [20]. We conduct a similar linear probing analysis on baselines combined with the BFP, and we additionally test the effect of our method on the naive FT baseline, which is denoted as **FT w/ BFP** in Figure 3. In FT w/ BFP, we do not use the memory buffer and thus $\alpha = \beta = 0$ in Equation 3, but we apply the BFP loss on the online data stream with $\gamma = 1$. Linear probing accuracies on Split CIFAR-10 are reported in Figure 3, where we also vary the portion of training data used for linear probing. The results show that BFP boosts the linear probing accuracies of the FT baseline by a significant margin, achieving a similar performance with the powerful experience replay method, DER++. When combined with DER++, BFP also helps improve the linear probing accuracies. This indicates that either with or without a memory buffer, BFP helps learn a better feature space during CL, where examples from different classes remain linearly separable.

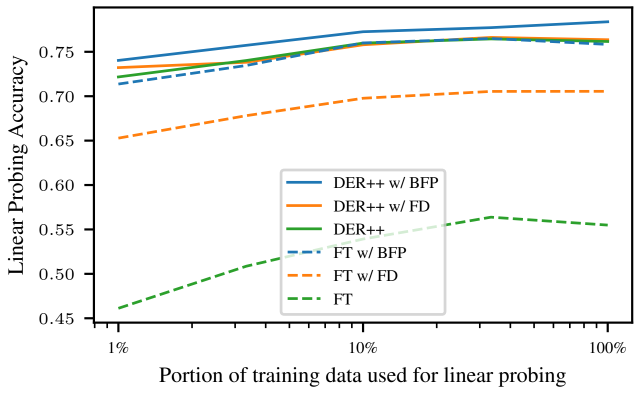

Figure 3. Linear probing accuracies on the frozen feature extractor that is obtained after training on Split-CIFAR10 with 200 buffer size, using different methods. A higher linear probing accuracy indicates a better feature space where data points are more linearly separable. Note that with the help of BFP, even a simple FT baseline can learn a representation as good as the powerful DER++ method.

Dataset	Buffer	FD	BFP	BFP-2
S-CIFAR10	200	68.44±1.35	**72.21±0.22**	72.04±0.96
	500	74.50±0.41	76.02±0.79	**76.63±0.63**
S-CIFAR100	500	43.81±1.35	**47.45±1.30**	**47.45±1.08**
	2000	56.56±0.55	**57.91±0.66**	57.27±0.67
S-TinyImg	4000	42.40±1.02	**43.40±0.41**	42.91±0.50

Table 2. Class-IL Final Average Accuracy using different types of layer for backward feature projection. The baseline method is DER++.

4.5. Ablation Study

We study the effect of different types of projection layers used for backward feature projection, in Equation 3. Our main results are obtained using a linear projection layer as

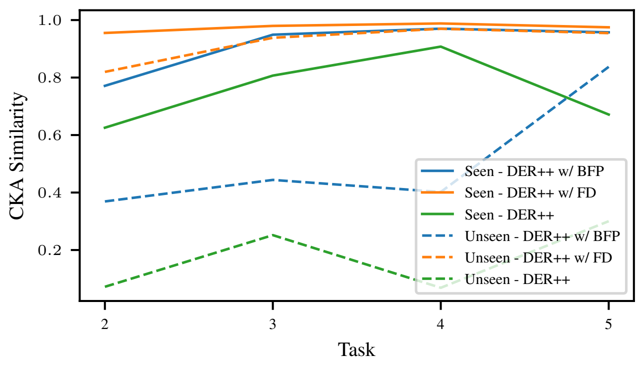

Figure 4. Feature similarity at different tasks of training on Split-CIFAR10, with 200 buffer size, using different CL methods.

the learnable transformation layer (denoted as **BFP**). We also test our method using an identity function $A = I$ as the projector, which is essentially a feature distillation loss (**FD**) on the final feature, as well as using a non-linear function (a two-layer MLP with ReLU activation in between) as p, which is denoted as **BFP-2**. These variations are tested when integrated with DER++ [8] and the results are shown in Table 2. According to Table 2, while the simple FD method already outperforms the baseline, the proposed learnable BFP further boosts the accuracies by a large margin. This is expected because FD regularizes the learned features directly to those from the old model, while the old model has not learned from the new data and may give useless features. In this case, FD promotes stability while lacking plasticity. On the contrary, BFP is learnable and thus provides the desired plasticity that allows new knowledge to appear while maintaining the old ones. Furthermore, we can also see that the performance already saturates with a linear projection layer and a more complex non-linear projection (BFP-2) does not improve further. We hypothesize that because BFP is applied on the feature space just before the linear classifier, linear separability is better maintained with a linear transformation rather than a non-linear function.

4.6. Feature Similarity Analysis

To demonstrate that the proposed BFP method regularizes the features that are already learned while allowing features of new data to freely evolve, we conduct an analysis of feature similarity. Following [15, 40], we adopt Centered Kernal Alignment (CKA) [28] to measure the feature similarity before and after training on a task. CKA is a similarity measure for deep learned representations, and it's invariant to isotropic scaling and orthogonal transformation [28]. CKA between two feature matrices $Z_1 \in \mathbb{R}^{d_1 \times n}$ and $Z_2 \in \mathbb{R}^{d_2 \times n}$ with a linear kernel is defined as

$$\text{CKA}(Z_1, Z_2) = \frac{\|Z_1 Z_2^T\|_F^2}{\|Z_1 Z_1^T\|_F^2 \|Z_2 Z_2^T\|_F^2}. \tag{15}$$

Recall that the feature matrix extracted from $\mathcal{D}_i$ using model h^j is denoted as $Z_i^j = h^j(\mathcal{D}_i)$, and similar $Z_{1:i}^j = h^i(\mathcal{D}_{1:i})$. During learning on task t, we consider two sets of features, features from $\mathcal{D}_{1:t-1}$ that have been learned by the model (seen) and features from $\mathcal{D}_t$ that are new to the model (unseen). We define their CKA similarity before and after learning on task t respectively as follows

$$\text{CKA}_t^{\text{seen}} = \text{CKA}(Z_{1:t-1}^{t-1}, Z_{1:t-1}^{t}) \tag{16}$$

$$\text{CKA}_t^{\text{unseen}} = \text{CKA}(Z_t^{t-1}, Z_t^{t}). \tag{17}$$

Note that Z_t^{t-1} represents the features extracted from future data by the old model, and it's expected that they do not provide useful information. On the contrary, $Z_{1:t-1}^{t-1}$ has already been well learned and we want to preserve its structure. Therefore, we want $\text{CKA}_t^{\text{seen}}$ to be high to retain knowledge, while $\text{CKA}_t^{\text{unseen}}$ low to allow the feature of unseen data to change freely in continual learning. We plot $\text{CKA}_t^{\text{seen}}$ and $\text{CKA}_t^{\text{unseen}}$ during CL in Figure 4 and the results confirm our desire. DER++ applies no direct constraint on the feature space during CL and thus similarity is low for both seen and unseen data. On the contrary, FD poses a strong constraint on both seen and unseen data, resulting in high similarities. In this way, FD gains more stability at the cost of lower plasticity. Combining their respective advantages, BFP keeps a high $\text{CKA}_t^{\text{seen}}$ while allowing the unseen features to change (low $\text{CKA}_t^{\text{unseen}}$), and thus is able to achieve a better trade-off between stability and plasticity.

5. Conclusion

In this paper, we reduce catastrophic forgetting in continual learning (CL) by proposing Backward Feature Projection (BFP), a learnable feature distillation method. We show that during CL, despite the large dimension of the feature space, only a small number of feature directions are used for classification. Without regularization, previously learned feature directions diminish and harm linear separability, resulting in catastrophic forgetting. The proposed BFP helps maintain linear separability learned from old tasks while allowing new feature directions to be learned for new tasks. In this way, BFP achieves a better trade-off between plasticity and stability. BFP can be combined with existing experience replay methods and experiments show that it can boost performance by a significant margin. We also show that BFP results in a more linearly separable feature space, on which a linear classifier can recover higher accuracies.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware

synapses: Learning what (not) to forget. In *ECCV*, 2018. 2

[2] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In *NeurIPS*, 2019. 2

[3] Elahe Arani, Fahad Sarfraz, and Bahram Zonooz. Learning fast, learning slow: A general continual learning method based on complementary learning system. In *ICLR*, 2022. 5, 6

[4] Tommaso Barletti, Niccoló Biondi, Federico Pernici, Matteo Bruni, and Alberto Del Bimbo. Contrastive supervised distillation for continual representation learning. In *ICIAP*, pages 597–609. Springer, 2022. 1

[5] Ari S. Benjamin, David Rolnick, and Konrad P. Körding. Measuring and regularizing networks in function space. In *ICLR*, 2019. 6, 7, 13

[6] Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-Jussa. Continual lifelong learning in natural language processing: A survey. *arXiv preprint arXiv:2012.09823*, 2020. 1

[7] Matteo Boschini, Lorenzo Bonicelli, Pietro Buzzega, Angelo Porrello, and Simone Calderara. Class-incremental continual learning into the extended der-verse. *arXiv preprint arXiv:2201.00766*, 2022. 5, 6, 11

[8] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *NeurIPS*, 2020. 1, 2, 3, 5, 6, 8, 11, 13

[9] Pietro Buzzega, Matteo Boschini, Angelo Porrello, and Simone Calderara. Rethinking experience replay: a bag of tricks for continual learning. In *ICPR*, 2021. 6, 11

[10] Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. New insights on reducing abrupt representation change in online continual learning. In *ICLR*, 2022. 3, 6, 7, 13

[11] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *ICCV*, 2021. 1

[12] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: understanding forgetting and intransigence. In *ECCV*, 2018. 2

[13] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019. 2

[14] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. In *ICML*, 2019. 5

[15] MohammadReza Davari, Nader Asadi, Sudhir Mudur, Rahaf Aljundi, and Eugene Belilovsky. Probing representation forgetting in supervised and unsupervised continual learning. In *CVPR*, 2022. 7, 8

[16] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *CVPR*, 2019. 1, 3

[17] Thang Doan, Mehdi Abbana Bennani, Bogdan Mazoure, Guillaume Rabusseau, and Pierre Alquier. A theoretical analysis of catastrophic forgetting through the ntk overlap matrix. In *AISTATS*, pages 1072–1080. PMLR, 2021. 14

[18] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *ECCV*, pages 86–102. Springer, 2020. 1, 3

[19] Laura N Driscoll, Lea Duncker, and Christopher D Harvey. Representational drift: Emerging theories for continual learning and experimental future directions. *Current Opinion in Neurobiology*, 76:102609, 2022. 3

[20] Enrico Fini, Victor G Turrisi da Costa, Xavier Alameda-Pineda, Elisa Ricci, Karteek Alahari, and Julien Mairal. Self-supervised models are continual learners. In *CVPR*, 2022. 3, 4, 7

[21] Alex Gomez-Villa, Bartlomiej Twardowski, Lu Yu, Andrew D Bagdanov, and Joost van de Weijer. Continually learning self-supervised representations with projected functional regularization. In *CVPR Workshop*, 2022. 3, 4

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6

[23] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *ICCV*, 2019. 6, 7, 13

[24] Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. Towards continual knowledge learning of language models. *arXiv preprint arXiv:2110.03215*, 2021. 1

[25] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016. 3

[26] Minsoo Kang, Jaeyoo Park, and Bohyung Han. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *CVPR*, 2022. 1, 3

[27] 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. 2

[28] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, pages 3519–3529. PMLR, 2019. 8

[29] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 5

[30] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NeurIPS*, 2017. 1, 14

[31] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 2020. 1

[32] Zhizhong Li and Derek Hoiem. Learning without forgetting. *PAMI*, 2017. 1, 2

[33] Xialei Liu, Chenshen Wu, Mikel Menta, Luis Herranz, Bogdan Raducanu, Andrew D Bagdanov, Shangling Jui, and Joost van de Weijer. Generative feature replay for class-incremental learning. In *CVPR Workshops*, 2020. 2

[34] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, 2017. 1, 2

[35] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018. 1

[36] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification. *arXiv preprint arXiv:2010.15277*, 2020. 5

[37] Marc Masana, Xialei Liu, Bartłomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification. *PAMI*, 2022. 1

[38] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989. 1

[39] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901. 4

[40] Vinay V Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020. 8

[41] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017. 2, 6, 13

[42] 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. 6, 13, 14

[43] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv*, 2016. 1

[44] Gobinda Saha, Isha Garg, Aayush Ankit, and Kaushik Roy. Space: Structured compression and sharing of representational space for continual learning. *IEEE Access*, 9:150480–150494, 2021. 14

[45] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NeurIPS*, 2017. 2

[46] Christian Simon, Piotr Koniusz, and Mehrtash Harandi. On learning the geodesic path for incremental learning. In *CVPR*, pages 1591–1600, 2021. 14

[47] Stanford. Tiny imagenet challenge, cs231n course., CS231N. 5

[48] Gido M Van de Ven and Andreas S Tolias. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018. 2

[49] Jianren Wang, Xin Wang, Yue Shang-Guan, and Abhinav Gupta. Wanderlust: Online continual object detection in the real world. In *ICCV*, pages 10829–10838, 2021. 1

[50] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, 2019. 5, 6, 7, 13

[51] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, pages 3014–3023, 2021. 1

[52] Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. In *CVPR*, 2020. 3

[53] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017. 1, 2

[54] Jingxin Zhang, Donghua Zhou, and Maoyin Chen. Monitoring multimode processes: A modified pca algorithm with continual learning ability. *Journal of Process Control*, 103:76–86, 2021. 14

[55] Xiao Zhang, Dejing Dou, and Ji Wu. Feature forgetting in continual representation learning. *arXiv preprint arXiv:2205.13359*, 2022. 7

[56] Fei Zhu, Zhen Cheng, Xu-Yao Zhang, and Cheng-lin Liu. Class-incremental learning via dual augmentation. *NeurIPS*, 34:14306–14318, 2021. 14

A. Acknowledgements

The authors thank Jongseong Jang, Yizhou (Philip) Huang, Kevin Xie and Nikita Dhawan for discussions and useful feedback.

B. Implementation Details

B.1. Complete Algorithm

In Algorithm 1, we provide the pseudocode of continual learning with the proposed BFP method. Note that following [8], we sample training datapoints from the memory buffer for each loss independently. We empirically find this results in better performance than using the same set of replayed samples for all losses. The images without augmentation x_o are pushed into the memory and replayed images are augmented on the fly. The classification model is trained using an SGD optimizer (*sgd*) and the projection matrix A is trained using an SGD+Momentum optimizer (*sgdm*).

B.2. Class-balanced Reservoir Sampling

We adopt the class-balanced reservoir sampling (BRS) [9] for memory buffer management. The detail of this algorithm is described in Algorithm 2. Compared to regular Reservoir Sampling (RS), BRS ensures that every class has an equal number of examples stored in the memory buffer. All experiments are incorporated with this change. Empirically we find that BRS does not bring significant changes compared to RS, but it helps to reduce variance in the results.

Algorithm 1 - Continual Learning with BFP

```
Input: dataset $\{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$, parameters $\theta = \{\phi, \psi\}$, scalars
$\alpha$, $\beta$ and $\gamma$, optimizer sgd, sgdm,
$M \leftarrow \{\}$
for $t$ from $1$ to $T$ do
    $A \leftarrow \textit{random-init}()$
    $\textit{sgdm} \leftarrow \textit{reinit}(\textit{sgdm})$
    for $(x_o, y_o)$ in $\mathcal{D}_t$ do
        $x, y \leftarrow \textit{augment}(x_o, y_o)$
        $L \leftarrow \textit{cross-entropy}(y, f_\theta(x))$                    {Eq. 8}
        if $t > 1$ then
            $x, y \leftarrow \textit{augment}(\textit{sample}(M))$
            $L_{\text{rep-ce}} \leftarrow \textit{cross-entropy}(y, f_\theta(x))$       {Eq. 9}
            $x, y \leftarrow \textit{augment}(\textit{sample}(M))$
            $L_{\text{rep-logits}} \leftarrow \|f_\theta(x) - f_{\text{old}}(x)\|_2$          {Eq. 10}
            $x, y \leftarrow \textit{augment}(\textit{sample}(M))$
            $L_{\text{BFP}} \leftarrow \|Ah_\psi(x) - h_{\text{old}}(x)\|_2$              {Eq. 11}
            $L = L + L_{\text{rep-ce}} + L_{\text{rep-logits}} + L_{\text{BFP}}$          {Eq. 12}
        end if
        $\theta \leftarrow \textit{sgd}(\theta, \nabla_\theta L)$
        $A \leftarrow \textit{sgdm}(A, \nabla_A L)$
        $M \leftarrow \textit{balanced-reservoir}(M, (x_o, y_o))$          {Alg. 2}
    end for
    $f_{\text{old}} = \textit{freeze}(f_\theta)$
end for
```

Algorithm 2 Balanced Reservoir Sampling [9]

```
1:  Input: replay buffer $M$, exemplar $(x, y)$,
2:         number of seen examples $N$.
3:  if $|M| > N$ then
4:      $M[N] \leftarrow (x, y)$
5:  else
6:      $j \leftarrow \text{RandInt}([0, N])$
7:      if $j < |M|$ then
          Reservoir Sampling
8:          $\boxed{M[j] \leftarrow (x, y)}$
          Balanced Reservoir Sampling
9:          $\tilde{y} \leftarrow \text{argmax ClassCounts}(M, y)$
10:         $k \leftarrow \text{RandChoice}(\{\tilde{k}; M[\tilde{k}] = (x, y), y = \tilde{y}\})$
11:         $M[k] \leftarrow (x, y)$
12:     end if
13: end if
```

B.3. Training details

Image sizes are 32×32 in Split-CIFAR10 and Split-CIFAR100 and 64×64 in Split-TinyImageNet. All experiments use the same data augmentation procedure, applied on input from both the current task and the memory buffer independently. Data augmentation includes a full-size random crop with a padding of 4 pixels, a random horizontal flip, and normalization.

For all experiments involving BFP, the optimizer for the matrix A is an SGD+Momentum optimizer with a learning rate of 0.1 and momentum of 0.9. The weighting term γ in Equation 12 is 1. Empirically we find that the BFP performance is not sensitive to these hyperparameters, and we use this one set of hyperparameters for BFP loss in all experiments.

B.4. Hyperparameters

In this section, we list the best hyperparameters used for the compared baselines mentioned in Section 4.2 and their results are reported Table 1. These hyperparameters are adopted from [8] and [7], where they were selected by a hyperparameter search conducted on a held-out 10% training set for validation. Please refer to [7, 8] for further details.

The proposed BFP only introduced a single hyperparameter γ, which is set to a constant value of 1 throughout all experiments and does not need extra tuning. Other hyperparameters like α and β are inherited from ER and DER++ [8] and we simply adopt the same set of hyperparameters from [8]. We do not further tune or modify them.

B.4.1 Split CIFAR-10

FT: $lr = 0.1$
JT: $lr = 0.1$

Buffer size = 200

iCaRL: $lr = 0.1$, $wd = 10^{-5}$
FDR: $lr = 0.03$, $\alpha = 0.3$
LUCIR: $\lambda_{\text{base}} = 5$, $mom = 0.9$, $k = 2$, epoch$_{\text{fitting}} = 20$, $lr = 0.03$, lr$_{\text{fitting}} = 0.01$, $m = 0.5$
BiC: $\tau = 2$, epochs$_{\text{BiC}} = 250$, $lr = 0.03$
ER-ACE: $lr = 0.03$
ER: $lr = 0.1$
DER++: $lr = 0.03$, $\alpha = 0.1$, $\beta = 0.5$

Buffer size = 500

iCaRL: $lr = 0.1$, $wd = 10^{-5}$
FDR: $lr = 0.03$, $\alpha = 1$
LUCIR: $\lambda_{\text{base}} = 5$, $mom = 0.9$, $k = 2$, epoch$_{\text{fitting}} = 20$, $lr = 0.03$, lr$_{\text{fitting}} = 0.01$, $m = 0.5$
BiC: $\tau = 2$, epochs$_{\text{BiC}} = 250$, $lr = 0.03$
ER-ACE: $lr = 0.03$
ER: $lr = 0.1$
DER++: $lr = 0.03$, $\alpha = 0.2$, $\beta = 0.5$

B.4.2 Split CIFAR-100

FT: $lr = 0.03$
JT: $lr = 0.03$

Buffer size = 500

iCaRL: $lr = 0.3$, $wd = 10^{-5}$
FDR: $lr = 0.03$, $\alpha = 0.3$
LUCIR: $\lambda_{\text{base}} = 5$, $mom = 0.9$, $k = 2$, epoch$_{\text{fitting}} = 20$, $lr = 0.03$, lr$_{\text{fitting}} = 0.01$, $m = 0.5$
BiC: $\tau = 2$, epochs$_{\text{BiC}} = 250$, $lr = 0.03$
ER-ACE: $lr = 0.03$
ER: $lr = 0.1$
DER++: $lr = 0.03$, $\alpha = 0.2$, $\beta = 0.5$

Buffer size = 2000

iCaRL: $lr = 0.3$, $wd = 10^{-5}$
FDR: $lr = 0.03$, $\alpha = 1$
LUCIR: $\lambda_{\text{base}} = 5$, $mom = 0.9$, $k = 2$, epoch$_{\text{fitting}} = 20$, $lr = 0.03$, lr$_{\text{fitting}} = 0.01$, $m = 0.5$
BiC: $\tau = 2$, epochs$_{\text{BiC}} = 250$, $lr = 0.03$
ER-ACE: $lr = 0.03$
ER: $lr = 0.1$
DER++: $lr = 0.03$, $\alpha = 0.1$, $\beta = 0.5$

B.4.3 Split TinyImageNet

FT: $lr = 0.03$
JT: $lr = 0.03$

Buffer size = 4000

iCaRL: $lr = 0.03$, $wd = 10^{-5}$
FDR: $lr = 0.03$, $\alpha = 0.3$
LUCIR: λ_{base}=5, mom=0.9, k=2, epoch$_{\text{fitting}}$=20, lr=0.03, lr$_{\text{fitting}}$=0.01, m=0.5
BiC: $\tau = 2$, epochs$_{\text{BiC}} = 250$, $lr = 0.03$
ER-ACE: $lr = 0.03$
ER: $lr = 0.1$
DER++: $lr = 0.1$, $\alpha = 0.3$, $\beta = 0.8$

C. Additional results

C.1. Final Forgetting

Final Forgetting (FF) measures the performance drop between after learning on each task and the end of CL. A CL method with a lower FF has a better ability to retain knowledge during CL and thus better stability. However, higher stability may come with the price of plasticity, and we remind readers that the Final Average Accuracy (FAA) reported in Table 1 can better reflect the trade-off between stability and plasticity. The Final Forgetting for all baselines and our methods can be found in Table 4. As we can see from Table 4, in the class-IL setting, the proposed DER++ w/ BFP method helps reduce FF compared to the base DER++ method by 11% and 12% on S-CIFAR10 with 200 buffer size and S-CIFAR100 with 500 buffer size respectively. DER++ w/ BFP also achieves the lowest FF over all compared methods in the class-IL setting. Final Forgettings in the Task-IL setting are generally much lower than those from the Class-IL setting because Task-IL provides the oracle task identifiers during the testing time and thus becomes a much easier CL scenario. In this setting, the proposed BFP also brings large improvements over the base ER and DER++ methods.

Dataset	Buffer	FD	BFP	BFP-2
S-CIFAR10	200	55.10±1.85	**63.27±1.09**	60.61±2.72
	500	66.37±1.37	**71.51±1.58**	70.25±1.18
S-CIFAR100	500	20.02±0.09	**22.54±1.10**	21.25±0.73
	2000	36.81±0.71	38.92±1.94	**39.42±2.54**
S-TinyImg	4000	23.13±0.77	**26.33±0.68**	25.87±0.86

Table 3. Class-IL Final Average Accuracy using different types of layer for backward feature projection. The base method is ER.

C.2. Ablation Study based on Experience Replay

We conduct the same ablation study as that in Section 4.5, on different types of the projection layer used in ER w/ BFP. The results are reported in Table 3. From Table 3, we can draw the same conclusion as in Section 4.5. BFP uses learnable linear transformation when distilling features and thus results in better plasticity during CL compared to the simple FD method. Results show that BFP outperforms FD by a significant margin and has better performance than BFP-2 in most cases. This further shows that enforcing a

Setting	Method	S-CIFAR10		S-CIFAR100		S-TinyImageNet
	Buffer Size	200	500	500	2000	4000
Class-IL	Joint Training	-		-		-
	Finetune	96.44±0.28		89.54±0.16		78.54±0.45
	iCaRL [41]	27.75±0.82	25.31±4.35	30.13±0.28	24.72±0.66	16.82±0.51
	FDR [5]	76.08±4.87	83.16±4.72	73.71±0.68	60.90±1.41	57.01±0.59
	LUCIR [23]	46.36±3.17	29.11±0.63	53.24±0.56	34.16±1.19	25.50±1.86
	BiC [50]	44.36±2.73	20.88±2.17	51.86±1.57	41.42±1.61	61.67±1.42
	ER-ACE [10]	21.59±1.19	15.07±0.99	38.32±1.29	28.69±0.87	30.83±0.23
	ER [42]	42.19±5.19	26.64±6.33	47.62±33.70	44.03±18.79	49.61±16.47
	ER w/ BFP (Ours)	32.23±5.58 (-9.96)	22.67±6.64 (-3.97)	47.69±30.30 (-0.07)	37.49±18.06 (-6.54)	41.59±20.77 (-8.02)
	DER++ [8]	28.28±1.06	20.16±1.49	42.58±2.03	26.29±1.66	16.03±1.20
	DER++ w/ BFP (Ours)	**16.69±0.28** (-11.59)	**13.25±0.64** (-6.91)	**29.85±0.97** (-12.73)	**20.91±0.86** (-5.39)	**9.42±1.04** (-6.28)
Task-IL	Joint Training	-		-		-
	Finetune	39.72±6.27		60.46±2.74		67.04±1.27
	iCaRL [41]	4.29±1.00	1.91±2.12	3.67±0.40	1.82±0.32	3.56±0.46
	FDR [5]	7.03±1.38	4.47±0.45	16.63±0.20	9.17±0.33	13.73±0.30
	LUCIR [23]	2.83±0.99	2.04±0.27	**2.61±0.17**	**1.08±0.13**	4.95±0.61
	BiC [50]	**0.81±0.77**	**0.24±0.25**	3.95±0.35	2.36±0.40	7.08±3.74
	ER-ACE [10]	6.10±0.72	3.64±0.29	13.95±0.45	7.36±0.43	10.67±0.41
	ER [42]	5.71±0.60	3.54±1.15	11.55±6.31	6.12±2.49	11.77±2.06
	ER w/ BFP (Ours)	1.38±0.29 (-4.34)	0.77±0.38 (-2.77)	5.63±1.56 (-5.92)	2.95±0.75 (-3.16)	**3.31±1.19** (-8.46)
	DER++ [8]	3.88±0.51	1.65±0.17	11.68±0.55	4.80±0.45	6.73±0.41
	DER++ w/ BFP (Ours)	1.04±0.23 (-2.84)	0.53±0.23 (-1.12)	6.36±0.43 (-5.32)	3.26±0.15 (-1.54)	4.17±0.37 (-2.49)

Table 4. Final Forgetting (FF, in %, lower is better) in Class-IL and Task-IL setting of baselines and our methods on various datasets and buffer sizes. The green numbers in parentheses show the absolute improvements over the corresponding ER or DER++ baselines brought by BFP.

linear relationship between the new and old features could better preserve linear separability and result in less forgetting in CL.

C.3. Linear Probing

We conduct the same linear probing analysis as Section 4.4 Figure 3 on Split-CIFAR100 and Split-TinyImageNet, and the results are reported in Figure 5. On these two datasets, while FD and BFP result in similar linear probing performance when based on DER++, BFP still leads to better linear probing accuracies when based on FT, especially when a large subset of training data is used for linear probing. FT w/ BFP (without the memory buffer) has a similar or even better performance than DER++ (with the memory buffer). This shows that BFP help learns a better feature space from CL, where features from different class are more linearly separable.

C.4. Feature Similarity Analysis

We perform the same feature similarity analysis as Section 4.6 and Figure 4 on Split-CIFAR100 and Split-TinyImageNet, and the results are reported in Figure 6. From Figure 6, although the curves have high variance throughout continual learning, we can see that BFP has feature similarities that are higher than the DER++ baseline but lower than the naive FD, and thus achieve a better trade-off between stability and plasticity.

Method	DER++	w/ FD	w/ BFP	w/ BFP-2
Class-IL	49.20±1.99	51.89±3.42	**54.45**±0.86	52.88±1.86
Task-IL	69.01±2.01	71.23±2.80	**72.05**±1.04	70.56±1.47

Table 5. Final Average Accuracy on ImageNet-100. (mean±std over 3 runs)

C.5. Experiments on Split-ImageNet100

To demonstrate that the proposed BFP method scales to large datasets, we conduct experiments on ImageNet100 [23, 41]. We split ImageNet100 into 10 tasks with 10 classes per task and use a memory buffer of size 2000. The model is trained for 65 epochs on each task using an SGD optimizer with an initial learning rate of 0.1 and weight decay of 5×10^{-4}. Within each task, the learning rate goes through a linear warm-up scheduler for the first 5 epochs and then decays with a 0.1 rate after 15, 30, 40, and 45 epochs. The results are reported in Table 5, which shows that the proposed BFP method still gives a significant improvement (over 5% in Class-IL setting) over the DER++ baseline, confirming our existing results.

C.6. Effect of γ on Plasticity and Stability

In continual learning, the weight of regularization loss controls how closely and strictly the model should resemble the old checkpoints. Therefore the weight serves as a control knob on the trade-off between stability and plasticity: with a stronger regularization loss, the model forgets old tasks less but instead has a hard time learning new tasks.

Although we did not perform an extensive hyperparameter search on γ for individual combinations of datasets and buffer sizes, we are still interested in how the varying γ affects the trade-off between stability and plasticity in continual learning. Therefore, we train DER++ w/ BFP on S-CIFAR10 with different γ and report the performance in Table 6. Besides FAA and FF, we also report the Average Learning Accuracy (ALA) [42], which measures the learning ability on new tasks in continual learning and thus reflects the plasticity. Using the notation from Sec. 4.1, ALA is defined as

$$ALA = \frac{1}{T}\sum_{i=1}^{T} a_i^i. \tag{18}$$

From Table 6, we can see that the effect of γ aligns with our intuition. A higher γ poses a strong regularization on the feature space, resulting in a lower FF (more stable) but also a lower ALA (less plastic). Also, we can observe that the final performance (FAA) remains robust to the value of γ within a considerable range.

D. More Related Work

There has been some recent work that also employs PCA computation in continual learning. Note that the proposed BFP does not require PCA computation during training and the feature directions are learned implicitly when optimizing matrix A. However, to provide a complete understanding of the literature, we briefly review the related work that also uses PCA for continual learning.

Doan *et al.* [17] proposed PCA-OGD, which combines PCA analysis with Orthogonal Gradient Descent (OGD). PCA-OGD projects the gradients onto the residuals subspace to reduce the interference of gradient updates from the new tasks on the old tasks. Zhu *et al.* [56] decomposed the learned features during CL using PCA. They showed that feature directions with larger eigenvalues have larger similarities (corresponding angles) before and after learning a task. They proposed that these feature directions are more transferable and less forgettable. They showed that their dual augmentation method can encourage learned features to have more directions with large eigenvalues. GeoDL [46] constructs low-dimensional manifolds for the features extracted by the online model and the old checkpoints and performs knowledge distillation on the manifolds. PCA computation is explicitly conducted on the learned features for the manifold construction. SPACE [44] used PCA analysis for network compression and pruning in continual learning. Similar to our analysis, they use PCA to split the learned filters in a network into Core, which is important for the current task, and Residual, which can be compressed and freed up to learn future tasks. In their work, PCA computation is required during continual learning on every layer of the network in order to do pruning, This poses a significant computational overhead in CL compared to our BFP. Instead of applying PCA analysis in continual learning, Zhang *et al.* [54] designed a modified PCA algorithm based on EWC [30] so that it has continual learning ability. They aim to reduce the forgetting problem in monitoring multimode processes.

γ	0.1	0.3	1.0	3.0	10.0
FAA	74.56	75.77	76.68	76.00	73.54
FF	16.11	14.63	13.16	13.07	12.69
ALA	87.45	87.32	87.21	86.45	83.70

Table 6. Results on CIFAR10 (buffer size 500) with different γ.

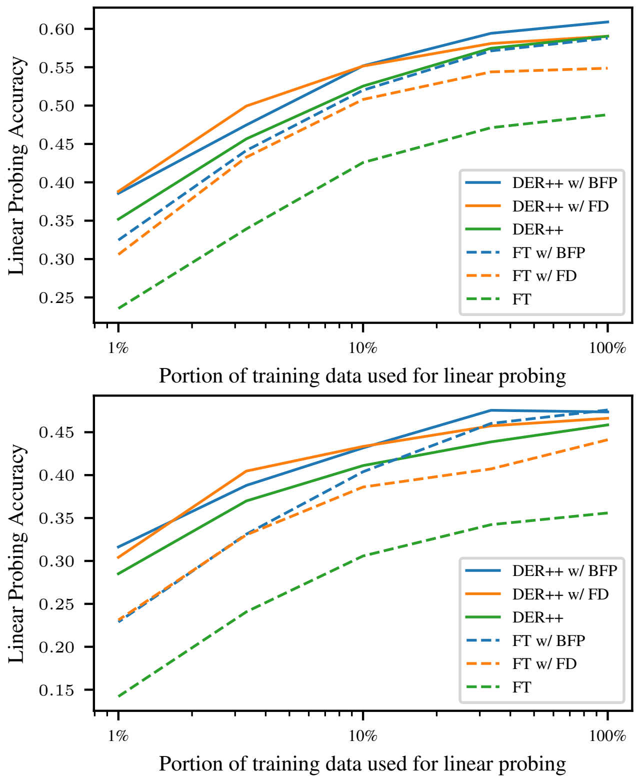

Figure 5. Linear probing accuracies on the fixed feature extractor obtained after training on Split-CIFAR100 (top) and TinyImageNet (bottom). DER++ and its variants use a buffer size of 500 for CIFAR100 and 4000 for TinyImageNet.

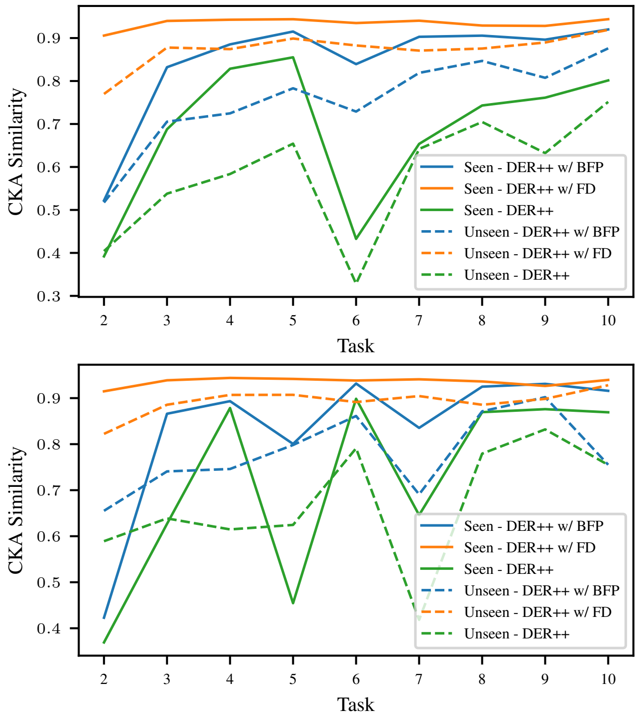

Figure 6. Feature similarity at different tasks of training on Split-CIFAR100 with buffer size 500 (top) and Split-TinyImageNet with buffer size 4000 (bottom), using different CL methods.