Navigating Memory Construction by Global Pseudo-Task Simulation for Continual Learning

Yejia Liu† **Wang Zhu**‡ **Shaolei Ren**†

†University of California Riverside ‡University of Southern California
{yliu807, shaolei}@ucr.edu wangzhu@usc.edu

Abstract

Continual learning faces a crucial challenge of catastrophic forgetting. To address this challenge, experience replay (ER) that maintains a tiny subset of samples from previous tasks has been commonly used. Existing ER works usually focus on refining the learning objective for each task with a static memory construction policy. In this paper, we formulate the dynamic memory construction in ER as a combinatorial optimization problem, which aims at directly minimizing the global loss across all experienced tasks. We first apply three tactics to solve the problem in the offline setting as a starting point. To provide an approximate solution to this problem in the online continual learning setting, we further propose the Global Pseudo-task Simulation (GPS), which mimics future catastrophic forgetting of the current task by permutation. Our empirical results and analyses suggest that the GPS consistently improves accuracy across four commonly used vision benchmarks. We have also shown that our GPS can serve as the unified framework for integrating various memory construction policies in existing ER works.

1 Introduction

Data in real world is non-stationary and keeps changing. Adapting new knowledge while maintaining the old skills learned from previous data is an idealism for intelligent systems. However, deep artificial neural networks suffer from catastrophic forgetting of old behaviors as the learning of new tasks keeps overwriting the past [30, 32, 33, 41, 34]. To combat this issue, numerous novel algorithms have been designed in continual learning in recent years [37, 14, 28, 9, 20, 3, 27]. Amongst them, the experience replay (ER) methods have been attested as one of the few methods that consistently achieve strong results across different continual learning setups [7, 12, 46, 21]. In the ER, a slightly relieved continual learning setting is considered, where a learner stores a subset of old examples from previous tasks in a fixed-sized memory and jointly trains the memory with the upcoming task.

Most existing ER-based works put rigorous efforts in refining the local learning objective to update the model parameters on one task at a time. In the methods, they stick to a single static memory construction policy, which is prone to failing in the long task sequence. For example, selecting random data examples from experienced tasks for the memory buffer can effect good generalization at the very beginning, but the forgetting rate would soon climb up when the tasks sequence becomes longer, as some class representations are totally squeezed out from the memory [12]. This major drawback has inspired us to explore the optimal memory construction with a dynamic policy.

In this work, we formulate the memory construction problem in ER as a combinatorial optimization problem. Unlike previous memory construction methods [2, 4, 16], we explicitly optimize the global objective, *i.e.*, the minimum loss of the final model on all observed tasks, by finding the best memory configuration strategy. As a starting point, we approach this problem in an offline setting, where we can go through the task sequence for multiple trials independently. By utilizing three tactics, we reduce the intractable search space to a significantly smaller one, then we use the binary search to solve the simplified problem in $O(T \log |\mathcal{M}|)$, where T is the total number of tasks and $|\mathcal{M}|$ is the size of the memory buffer. Based on the offline solution, in the online continual learning setup,[1] we propose our method **Global Pseudo-task Simulation (GPS)** as an approximate solution to the problem. The GPS mimics the catastrophic forgetting pattern for the current task by creating future pseudo-tasks. We examine a few simulation methods and find permutation is the favorable way to synthesize pseudo-tasks.

We conduct experiments on four widely used vision benchmarks to show that GPS achieves higher accuracy compared to baselines, especially when we have a long task sequence. Besides, GPS, as a solution for dynamic memory construction, can also be easily applied to other ER variants [7, 10] to further improve their performance.

2 Preliminary and Notations

Continual Learning In continual learning, the model $f(\theta)$ experiences a stream of data points $(x_i, y_i) \sim P_i$ from a sequence of tasks t_i, where $i \in \mathcal{T} = \{1, ..., T\}$, and P_i is an unknown i.i.d. distribution of task t_i. Without experience replay, the model $f(\theta)$ is optimized on one task at a time following the task sequence under the tight constraint that the examples from previous tasks cannot be accessed [38]. We denote θ_i as the parameter of $f(\cdot)$ *after* training task t_i, and we refer to the function $g(\cdot)$ that updates θ_i for each task t_i as the *local updating method.* Once the local updating method is determined, θ_T can be derived recursively by $\theta_i = g(\theta_{i-1}, P_i)$ from θ_0, which is the initialization point.

After sequentially training T tasks, the objective of continual learning is to achieve the minimum loss across all observed tasks with the final model in the end. We write the summed *global loss* $\mathcal{L}^G$ as in Eqn. (1), where θ_T is the final parameter after T tasks and $l(\cdot)$ is the cross-entropy loss. For convenience, we also denote the global loss for a single task i as $\mathcal{L}_i^G = \mathbb{E}_{(x_i,y_i)\sim P_i} \ell(y_i, f(x_i; \theta_T))$.

$$\mathcal{L}^G = \sum_{i=1}^{T} \mathbb{E}_{(x_i,y_i)\sim P_i} \ell(y_i, f(x_i; \theta_T)) \tag{1}$$

Experience Replay Previous works [14, 38, 46] have proposed a series of local updating methods to optimize the global objective. Amongst them, one effective way is experience replay (ER) from [12]. Experience replay relieves a bit on the tight constraint in continual learning by adding a fixed-sized memory buffer $\mathcal{M}$ to store a limited subset of seen examples.

We denote the memory *after* training task t_i as $\mathcal{M}_i$. The modified local updating method g treats the memory as another input, and jointly optimizes examples of the current task and examples stored in the memory, with a factor λ on the loss of memory examples as shown in Eqn. (2). Thus, θ_i is iteratively updated by $\theta_i = g(\theta_{i-1}, P_i, \mathcal{M}_{i-1})$.

$$g(\theta, P, \mathcal{M}) = \arg\min_{\theta} \{\mathcal{L}_t(\theta, P) + \lambda \mathcal{L}_t(\theta, \mathcal{M})\} \tag{2}$$

$$\mathcal{L}_t(\theta, P) = \mathbb{E}_{(x,y)\sim P} \ell(y, f(x; \theta)) \tag{3}$$

Both empirical results [12, 7] and theoretical analysis [21] have suggested that the local updating method g of experience replay is effective on reducing the global loss $\mathcal{L}^G$. If not specially specified, we refer to local updating method g as Eqn. (2) in the following text.

3 Problem Formulation: Dynamic Memory Construction

Most previous works based on ER regard the global loss $\mathcal{L}^G$ as a function of g with a static memory construction strategy for $\mathcal{M}$. They utilize various techniques like regularization [7, 10] or memory sampling [2] to further refine the local updating method g. Unlike them, we view the global loss $\mathcal{L}^G$ as a function of the memory $\mathcal{M}$, and explicitly minimize $\mathcal{L}^G$ by optimizing the memory *without* modifying the local updating methods in Eqn. (2).

[1]We use *online* as opposed to the multi-trial offline setup, while we still use the multi-pass training here.

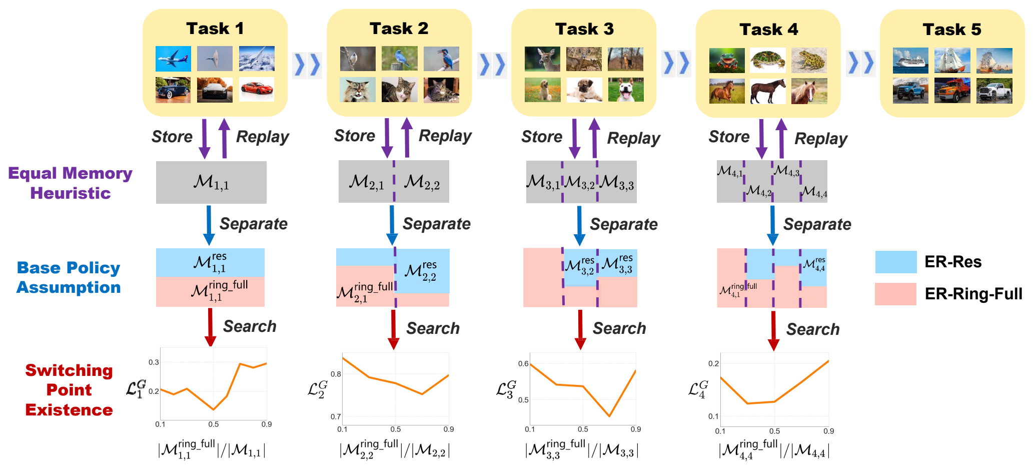


Figure 1: Using three tactics on the S-CIFAR-10 dataset to solve the dynamic memory construction problem in the offline setting. The equal memory heuristic and the base policy assumption reduce the search space. The switching point existence enables the binary search.

Following the setup of ER, we propose dynamic memory construction, which aims at finding the best memory construction $\mathcal{M}$ to optimize the global objective $\mathcal{L}^G$, as defined in Eqn. (1). This combinatorial optimization problem is expressed by the Eqn. (4), where we minimize the global objective by considering the memory $\mathcal{M}_i$ after task t_i as variables.

$$\min_{\{\mathcal{M}_i\}_{i\in\mathcal{T}}} \mathcal{L}^G(\{\mathcal{M}_i\}_{i\in\mathcal{T}}) \tag{4}$$

We denote the part in $\mathcal{M}_i$ storing the examples of task t_j as $\mathcal{M}_{i,j}$. There are four intrinsic constraints for the optimization problem. In all constraints, $i, j, i' \in \mathcal{T}$.

- $\mathcal{M}_i$ is the union of memories $\{\mathcal{M}_{i,j}\}_{j\in\mathcal{T}}$. The collection $\{\mathcal{M}_{i,j}\}_{j\in\mathcal{T}}$ is mutually disjoint.
- No future examples in the memory, *i.e.*, $\mathcal{M}_{i,j} = \varnothing$ when $i < j$.
- The size of memory is the same across the training procedure from t_1 to t_T, denoted as $|\mathcal{M}|$.
- As we cannot access previous examples that are not stored in the memory, once the set for task t_j in the memory is constructed after training task t_j, its size will not increase. Thus, the memory set for task j in $\mathcal{M}_i$ is always a subset of that in $\mathcal{M}_{i'}$ given $j \le i' < i$.

4 An Offline Solution: Reduce the Search Space

In order to reduce the intractable search space of $\mathcal{M}_\mathcal{T}$ quantitatively, we first consider the dynamic memory construction problem in an offline setup as the starting point for analyzing the realistic online setting. In the offline setup, we can go through the task sequence for multiple trials independently, but we are still strictly not allowed to access previous examples that are not in $\mathcal{M}$ in each trial. We take three tactics to reduce the search space, referred to as the *equal memory heuristic*, the *base policy assumption* and the *switching point existence*. Fig. 1 illustrates how we apply three tactics to the S-CIFAR-10 dataset. Based on these tactics, we provide an approximate solution to the problem.

4.1 Equal Memory Buffer for Each Task

We first take the equal memory heuristic from [12], which has shown that an equal size of memory buffer for each observed task is effective to avoid catastrophic forgetting.[2] Based on this tactic, we add $|\mathcal{M}_{i,j}| = |\mathcal{M}|/i$, where $i \ge j$, to the previous constraints list. Given a fixed size of each $\mathcal{M}_{i,j}$, instead of optimizing $\mathcal{M}_i$ from previous observed tasks jointly, we can independently optimize each

[2]For simiplicity of the formulation, we assume each task has the same number of classes in $\mathcal{M}$.

$\mathcal{M}_{i,j}$ and construct $\mathcal{M}_i$ from $\mathcal{M}_{i,j}$. To this end, we replace the optimization objective Eqn. (4) by a simplified version Eqn. (5), taking $\mathcal{M}_{i,j}$ as variables.

$$\min_{\{\mathcal{M}_{i,j}\}_{i,j\in\mathcal{T}}} \mathcal{L}^G(\{\mathcal{M}_{i,j}\}_{i,j\in\mathcal{T}}) \tag{5}$$

However, given the equal memory heuristic, the search space in Eqn. (5) is still very large. For each task t_j, suppose the total number of examples for task t_j is n_j and $n_j > |\mathcal{M}|$, $\mathcal{M}_{i,j}$ has more than $\binom{|\mathcal{M}|}{|\mathcal{M}|/i}$ different constructions.

4.2 Mix of Base Policies

To further reduce the search space in Eqn. (5), we introduce two base policies, ER-Res and ER-Ring-Full [12]. For a given task t_i, ER-Res builds a memory by random samples, while ER-Ring-Full builds a memory by selecting the same number of samples from each class.

We take the base policy assumption, where we deem that the memories having the same size and taking the same base policies are the same. It implies that even though two memory buffers contain different data examples, they are still identified as the same as long as they meet these two rules. This assumption is backed up by [7] where the standard deviation of accuracy on different benchmarking datasets is quite small (∼0.5) when using the same policy with a relatively large $|\mathcal{M}|$. Based on the assumption, we separate $\mathcal{M}_{i,j}$ into two disjoint parts and take the mixed policy, where $\mathcal{M}^{\text{res}}_{i,j}$ and $\mathcal{M}^{\text{ring-full}}_{i,j}$ represent two parts in $\mathcal{M}_{i,j}$, namely the former taking the ER-Res and the latter taking ER-Ring-Full policies, respectively. For completeness, we extend the subset constraint to $\mathcal{M}^{\text{res}}_{i,j} \subseteq \mathcal{M}^{\text{res}}_{i',j}$, $\mathcal{M}^{\text{ring-full}}_{i,j} \subseteq \mathcal{M}^{\text{ring-full}}_{i',j}$, where $j \le i' < i$.

Previous studies of ER [12] have shown the randomness (ER-Res) is crucial in a large memory, while guaranteeing the equal representation of each class (ER-Ring-Full) is crucial under a tiny memory. As the number of tasks grows, the size of memory for each task t_i becomes smaller. Under a mixed policy, it is natural to first shrink the ER-Res part of the memory and then shrink the ER-Ring-Full part of the memory. By doing so, given $\mathcal{M}^{\text{res}}_{j,j}$ and $\mathcal{M}^{\text{ring-full}}_{j,j}$, we can determine $\mathcal{M}^{\text{res}}_{i,j}$ and $\mathcal{M}^{\text{ring-full}}_{i,j}$ accordingly for each $i > j$. We denote the size of $\mathcal{M}^{\text{ring-full}}_{j,j}$ as a_j and derive the size of $\mathcal{M}^{\text{res}}_{j,j}$ as $(|\mathcal{M}|/j) - a_j$. Substituting the collection of sets $\{\mathcal{M}_{i,j}\}_{i,j\in\mathcal{T}}$ by an integer variable a_j, we therefore further simplify the optimization objective, as in Eqn. (6).

$$\min_{\{a_j\}_{j\in\mathcal{T}}} \mathcal{L}^G(\{a_j\}_{j\in\mathcal{T}}) \tag{6}$$

where each a_j has $|\mathcal{M}|/j$ different choices, significantly smaller than the search space for $\mathcal{M}_{i,j}$ in Eqn. (5). However, the total search space equals to $|\mathcal{M}|^{T-1}/(T-1)!$, which is still large.

4.3 Existence of a Switching Point

For the memory allocation after each task t_j, there exists a gold point $a_j = s_j$ that assigns exactly the *required* ring-full memory to keep the best balance of $\mathcal{M}^{\text{ring-full}}_{i,j}$ and $\mathcal{M}^{\text{res}}_{i,j}$ for the following task t_i. If the ring-full memory size is not large enough, we lose the power of forcing an equal representation of classes as the task sequence grows. Conversely, if the ring-full memory is more than enough, we sacrifice the randomness when the task number is still small. To this end, we assume s_j is a unique switching point, *i.e.*, there exists a switching point s_j for each a_j, which satisfies the monotonicity

$$\mathcal{L}^G(\{a'_j\} \cup \{a_i\}_{i\in\mathcal{T}/\{j\}}) > \mathcal{L}^G(\{a_j\} \cup \{a_i\}_{i\in\mathcal{T}/\{j\}}), \quad a'_j < a_j < s_j \text{ or } s_j < a_j < a'_j \tag{7}$$

Notice that the global loss of task t_j should depend closely on what data we store and replay for task t_j, but very loosely on what data we store and replay for other tasks. Following this intuition, though it is hard to verify a specific point satisfies the condition of Eqn. (7) for all combinations of $\{a_i\}_{i\in\mathcal{T}/\{j\}}$, we can simplify the switching point condition to

$$\mathcal{L}^G_j(a'_j) > \mathcal{L}^G_j(a_j), \qquad a'_j < a_j < s_j \text{ or } s_j < a_j < a'_j \tag{8}$$

Given the switching point existence, though we still have the same search space, instead of a linear search, we can apply a binary search algorithm to reduce the time complexity to $O(T\log|\mathcal{M}|)$. Note that if we search by comparing against the exact left and right integer of the point, *i.e.*, comparing the loss computed by $a_j + 1, a_j - 1$ and a_j, the variance of sampling may cause the algorithm to exit unexpectedly. To increase the robustness of the algorithm, we take a search stride ϵ, where we compare the loss computed by $a_j + \epsilon, a_j - \epsilon$ and a_j. The stride is determined by the benchmark and the size $|\mathcal{M}_j|$. The detailed binary search algorithm for s_j can be found in our Appendix A.

This assumption is valid for over 85% of the cases from the observations in Fig. 1 (for the S-CIFAR-10 dataset) and Appendix B (for other benchmarks). For each a_j (*i.e.*, $|\mathcal{M}_{j,j}^{\text{ring-full}}|$) in the figures, the global loss $\mathcal{L}^G$ initially decreases monotonically and then increases monotonically as a_j grows. Rarely but possible, due to the variance of sampling, we cannot find the switching point s_j. In such cases, we will choose a_j with the minimum loss amongst the values we searched.

5 Global Pseudo-task Simulation (GPS)

The offline solution of dynamic memory construction requires going through the task sequence for $O(T \log |\mathcal{M}|)$ times to find $\{s_j\}_{j \in \mathcal{T}}$, which violates the online continual learning setup. To circumvent this issue, we propose Global Pseudo-task Simulation (GPS), which provides an approximate solution $\{\tilde{s}_j\}_{j \in \mathcal{T}}$ to the problem by simulating the future training process under the online setup. Specifically, we simulate the local updating process Eqn. (2) by creating *pseudo-future tasks*.

5.1 Objective Function for Simulation

Perfectly simulating the future is a mission impossible in continual learning [21]. As we have no information about the future tasks, the distribution of the pseudo-future tasks we create could be quite different from the distribution of the real future ones. To find each approximated switching point $\tilde{s}_j$ more precisely, we intend to use more real tasks and less pseudo-future tasks as possible. As we are required to allocate examples of task t_j to a non-empty set $\mathcal{M}_{j,j}$ right after training the task t_j, we solve $\tilde{s_j}$ by a simulation process from θ_j to θ_T, without future modifications.

To this end, we modify the offline objective function Eqn. (6) to the online simulation objective Eqn. (9), where $\tilde{\theta}_{j:i}$ is the simulated θ_i from the real θ_j, for $i > j$.

$$\tilde{s}_j = \arg\min_{a_j} \mathbb{E}_{(x_j, y_j) \sim P_j} \ell(y_j, f(x_j; \tilde{\theta}_{j:T})) \tag{9}$$

Note that $\tilde{\theta}_{j:T}$ is derived recursively from $\tilde{\theta}_{j:i} = g(\tilde{\theta}_{j:(i-1)}, \tilde{P}_i, \tilde{\mathcal{M}_{i-1}})$, initialized with $\tilde{\theta}_{j:j} = \theta_j$. $\tilde{P}_i$ is the task distribution of the synthesized pseudo-tasks $\tilde{t}_i$, and $\tilde{\mathcal{M}}_i$ is the simulated pseudo-memory after training the pseudo-task $\tilde{t}_i$. We restrict the global objective to the evaluation of task t_j, as we cannot access the previous tasks as well as the future tasks. We show in Appendix C that the summed global loss $\mathcal{L}^G$ correlates with the global loss of a single task t_j positively as a function of a_j.

5.2 Synthesizing Pseudo-tasks

The goal of our pseudo-task simulation is to find s_j precisely, *i.e.*, $\tilde{s}_j \approx s_j$. Concretely, instead of accurately simulating the future training process, we use synthesized pseudo-tasks to *mimic the forgetting patterns of task t_j caused by the future tasks*. To achieve this aim, we believe if the real task sequence holds certain properties, it is essential for pseudo-tasks to hold the same set of properties to mimic the same forgetting pattern.

We find most of the existing widely used vision CL benchmarks [7, 25, 43] hold two properties: 1) similar learning difficulty of individual tasks; 2) limited zero-shot transfer ability. The experimental validation of these properties is in Appendix C.1. To induce the same level of catastrophic forgetting, we expect a pseudo-task $\tilde{t_j}$ to have similar difficulty as its real counterpart t_j in the future, measured by the accuracy in an identical end-to-end training setup [34]. To avoid the similarity between tasks to lead to little forgetting after training on pseudo-tasks, pseudo-tasks should also have low zero-shot accuracy with the model trained on the current task.

Based on these two properties, we synthesize pseudo-future tasks from the task t_j by applying different permuting seeds to its input x_j to create a series of future tasks $\{\tilde{t}_{j+1}, ..., \tilde{t}_T\}$. We achieve this by permuting the pixels of each image on image datasets, i.e., a fresh permutation would be generated and applied to all images within a synthesized task [54].

Besides permutation, we have also considered two other synthesizing techniques that lack a certain property we discussed for comparison. One is rotation, where we rotate the image inputs of the task t_j gradually by 15 degrees to create pseudo-future tasks [7]. The other is blurring, where we apply the Gaussian blurring into the image using a 5×5 filter with growing standard deviation. Rotation creates pseudo-tasks with similar difficulties as the real tasks, but the zero-shot transfer ability is far too good. Blurring creates pseudo-tasks with limited zero-shot transfer ability, yet the task difficulty is increasing along the pseudo-task sequence.

5.3 Construct Pseudo-memories

Fig. 2 visualizes the process of pseudo-task training and pseudo-memory construction. During the simulation process after task t_j, we try different a_j with a binary search and pick the best one as $\tilde{s}_j$ based on the online objective Eqn. (9).

We start from $\tilde{\mathcal{M}}_j$, which is the same as the real $\mathcal{M}_j$. The construction for $\tilde{\mathcal{M}}_i$ $(i > j)$ is different for real tasks and pseudo-tasks. For real tasks $t_{j'}, j' \in \{1, ..., j\}$, $\tilde{\mathcal{M}}_{i,j'}$ is the real $\mathcal{M}_{i,j'}$ determined for a given $a_{j'}$, as discussed in § 4.2. Note that the approximated switching points $\{\tilde{s}_{j'}\}_{j' \in \{1,...,j-1\}}$ for the previous $j-1$ tasks are solved before $\tilde{s}_j$. For pseudo-tasks $\tilde{t}_{j'}, j' \in \{j+1, ..., i\}$, $\tilde{\mathcal{M}}_{i,j'}$ is constructed by randomly selecting $|\mathcal{M}|/i$ pseudo-data from $\tilde{t}_{j'}$.

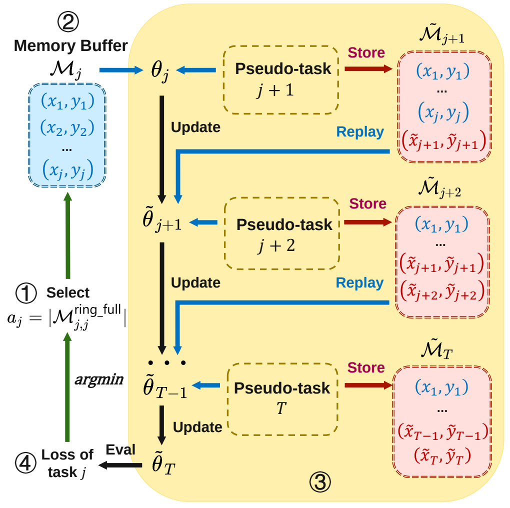


Figure 2: The Global Pseudo-task Simulation process to solve a_j after training task t_j.

5.4 Other Implementation Details

For the efficiency of simulation, the number of examples of the synthesized pseudo-tasks we use in the simulation is equal to $|\mathcal{M}|$. As the simulation quickly converges on a small number of examples, we train fewer epochs for each task compared with the real training process. Besides, when the number of tasks T are large or unknown, we extend GPS by simulating a fixed-sized sliding window of future tasks for the sequence, *e.g.*, simulating 10 pseudo-future tasks for each task. Also, during the local update of each task t_i, for a smooth transition, we allow the current training task t_i to take up to $|\mathcal{M}|/i$ the size of the memory without changing $\mathcal{M}_{i,j}$ for any previous task t_j.[3] We also put the detailed algorithm for GPS in Appendix A.

6 Experiments

6.1 Experimental Setup

Datasets We carry out evaluations on four widely used vision benchmarks in continual learning, **P-MNIST**, **S-CIFAR-10**, **S-CIFAR-100** and **TinyImageNet** [43, 7, 25]. P-MNIST was proposed in [18]. It contains 10 tasks where the first task is the MNIST dataset [23] while the later ones are constructed by permuting each image in MNIST with an unique permutation seed. The S-CIFAR-10 is constructed by splitting CIFAR-10 [1] into 5 sequential tasks where each task contain 2 classes and 12,000 images [22, 7]. Similarly, we split CIFAR-100 [1] into 10 tasks where each one contains 10 classes and 6, 000 images to construct S-CIFAR-100. The TinyImagenet [48] is a subset of ImageNet [15] with 200 classes. We split it into 10 consecutive tasks with 20 classes per task.

Architectures For P-MNIST, we apply a fully connected network with two hidden layers. Each comprises 100 ReLU units. For S-CIFAR-10, S-CIFAR-100 and TinyImageNet, we use Resnet18 following [7] and [14].

[3]We release our code at https://github.com/liuyejia/gps_cl

Table 1: Accuracy of GPS using different simulation techniques and baselines on four vision benchmarks. ER-Oracle shows the performance of the offline solution as described in § 4. Reported numbers are all averaged over 5 runs.

Method $\lvert\mathcal{M}\rvert$	Simulation	**P-MNIST** 1000	**S-CIFAR-10** 200	**S-CIFAR-100** 2000	**TinyImageNet** 2000
ER-Res	-	$86.55_{\pm 0.48}$	$92.01_{\pm 0.80}$	$81.38_{\pm 0.51}$	$57.50_{\pm 0.54}$
ER-Ring-Full	-	$84.33_{\pm 0.65}$	$91.53_{\pm 0.56}$	$81.16_{\pm 0.65}$	$54.73_{\pm 0.32}$
ER-Hybrid	-	$86.84_{\pm 0.35}$	$92.06_{\pm 0.89}$	$81.47_{\pm 0.23}$	$57.97_{\pm 0.44}$
GPS	**Permutation**	$\mathbf{87.93_{\pm 0.11}}$	$\mathbf{92.77_{\pm 0.39}}$	$\mathbf{82.46_{\pm 0.33}}$	$\mathbf{59.26_{\pm 0.23}}$
	Rotation	$85.38_{\pm 0.20}$	$91.61_{\pm 0.49}$	$81.50_{\pm 0.42}$	$57.45_{\pm 0.33}$
	Blurring	$86.03_{\pm 0.31}$	$91.96_{\pm 0.38}$	$81.49_{\pm 0.46}$	$56.85_{\pm 0.27}$
ER-Oracle	Offline	$88.26_{\pm 0.15}$	$93.09_{\pm 0.35}$	$82.88_{\pm 0.31}$	$60.56_{\pm 0.32}$

Baselines The baselines we used in experiments include ER-Res, ER-Ring-Full and ER-Hybrid [12]. ER-Hybrid is a mix of ER-Res and ER-Ring-Full, where the memory construction strategy would switch from the former to the latter once observing only one sample of some class is left in $\mathcal{M}$. We also compare to non-ER methods: online EWC (oEWC) [44], iCaRL [37], A-GEM [11] and GSS [4].

Training Details: Our training all use stochastic gradient descent (SGD) with a learning rate of 0.1. We use $\lambda = 1$ in the local updating method Eqn. (2). For P-MNIST, we train 5 epochs for each task while increasing the number of epochs to 50 for S-CIFAR-10, 100 for both S-CIFAR-100 and TinyImageNet regarding their data complexity, as done by works [7, 43]. For P-MNIST and S-CIFAR-10, we set the batch size as 10. For S-CIAFR-100 and TinyImageNet, batch size is set to 50. Following the implementation of ER [12], we set the same batch size for training the current task and the memory. Note the permutation seeds we use for simulation in P-MNIST is different from the ones used in P-MNIST itself to avoid peeping into test datasets. More training details and hyperparameter values can be found in the Appendix F.

6.2 Main Results

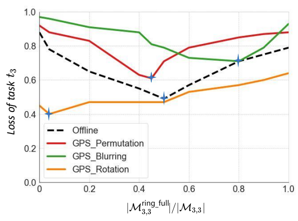
Our GPS using permutation shows improved accuracy compared to other baselines in Table 1. Its performance is close to the performance of the offline oracle solution, which implies it is a good approximation. We can also see that GPS using permutation performs better than using rotation and blurring, in terms of both accuracy and stability. The results support our hypotheses on the properties that synthesized tasks should bear, *i.e.*, similar level difficulties to previous tasks and limited zero-shot transfer ability. We compute the L1-norm of the offline sampling ratios *vs.* permuting-simulated sampling ratios, which are 0.9, 1.2, 1.2 for P-MNIST, S-CIFAR-100 and TinyImageNet, respectively. This shows the method selects sampling ratios for other benchmarks almost as good as that for the P-MNIST, which implies the similarity of pseudo-tasks is not the major factor for the enhanced results.

Besides, we notice that the GPS using different simulation techniques is more stable as it achieves lower standard deviation on almost every evaluation dataset. We attribute this to the reduced interference from random seeds as we take the global objective into explicit consideration. We have also included the experimental results of GPS with smaller memory in Appendix D.1.

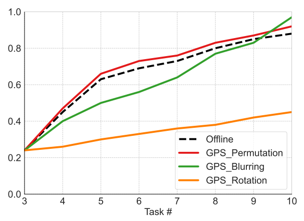
6.3 Analysis of Simulation Methods

To further understand why permutation works better than other simulation techniques, we analyze the task t_3 of the S-CIFAR-100 benchmark. We plot the curve of global loss w.r.t. the ratio of ER-Ring-Full in the memory $\mathcal{M}_{3,3}$, and the forgetting curve of task t_3 after training future tasks, as in Fig. 3.

We can see that the rotation induces less forgetting (lower loss) than the real tasks (offline cases) along the sequence, as shown in Fig. 3(b), as rotation creates tasks sequences that bear good zero-shot transfer from the previous tasks. The pseudo-task sequence created by rotation is therefore too easy such that the switching point of the curve is close to 0, as shown in the Fig. 3(a), which implies it is



(a)



(b)

Method	$T = 20$	$T = 40$
ER-Res	$73.25_{\pm 1.07}$	$47.33_{\pm 2.75}$
ER-Ring-Full	$73.92_{\pm 1.47}$	$49.26_{\pm 2.66}$
ER-Hybrid	$74.87_{\pm 0.88}$	$51.13_{\pm 2.61}$
GPS	$\mathbf{77.93}_{\pm \mathbf{0.20}}$	$\mathbf{56.57}_{\pm \mathbf{0.63}}$
ER-Oracle	$78.51_{\pm 0.17}$	$57.10_{\pm 0.51}$

(c)

Figure 3: (a). Global loss of task t_3 from the S-CIFAR-100 benchmark w.r.t. different configurations of $\mathcal{M}_{3,3}$. The blue stars mark the switching points. (b). The loss of task t_3 after training future tasks. (c). Accuracy of GPS and baselines in long task sequence on P-MNIST.

Table 2: (a). Time cost (in minutes) of training *vs*. simulation for P-MNIST and S-CIFAR-10. (b). Accuracy of GPS when incorporating existing ER variants, DER++ [7] and HAL[10], comparing to other methods. For these two ER variants, we use a memory size of 1000 on P-MNIST and a memory size of 2000 on the TinyImageNet.

Short Seq	# Tasks	Training	Simulation
P-MNIST	10	26.32	2.30
S-CIFAR-10	5	545.56	10.02
S-CIFAR-100	10	1187.93	137.63
TinyImageNet	10	2418.20	209.98
Long Seq	# Tasks	Training	Simulation
P-MNIST	20	55.29	6.21
P-MNIST	40	108.17	13.37

(a)

	P-MNIST	**TinyImageNet**
oEWC	$69.21_{\pm 2.92}$	$20.81_{\pm 0.95}$
iCaRL	-	$38.77_{\pm 3.68}$
GSS	$86.34_{\pm 4.28}$	-
A-GEM	$77.36_{\pm 1.28}$	$25.30_{\pm 0.87}$
OGD	$81.52_{\pm 2.21}$	-
HAL	$87.69_{\pm 0.34}$	-
GPS+HAL	$\mathbf{88.23}_{\pm \mathbf{0.03}}$	-
DER++	$91.14_{\pm 0.22}$	$60.67_{\pm 1.08}$
GPS+DER++	$\mathbf{91.64}_{\pm \mathbf{0.16}}$	$\mathbf{61.01}_{\pm \mathbf{0.98}}$

(b)

similar to an ER-Res static policy. As for the blurring, the forgetting curve first climbs slowly, as it allows some zero-shot transfer from the previous tasks. And then, the loss increases dramatically since the task difficulty goes up. Its pattern of forgetting is quite different from the real tasks (offline case) as shown in the Fig. 3(b). These empirical results have backed up our hypotheses in § 5.2.

We can also observe that the GPS using permutation has the closest switching point to the offline compared to the rest. Though permutation creates pseudo-task sequence that result in higher losses than the real task sequence, the pattern of forgetting is similar[4].

6.4 Long Task Sequence

We extend the 10-task P-MNIST benchmark to 20 and 40 tasks to create a longer task sequence. Fig. 3(c) implies the dynamic memory construction is even more crucial when the task sequence is long in continual learning. The offline solution, *i.e*., ER-Oracle, outperforms the baseline policies by more than 5% accuracy when $T = 40$. We attribute the performance gain to optimizing the global objective directly. From the Fig. 3(c), We can also see that GPS solves the problem significantly better than the baselines. Interestingly, the GPS still achieves close performance to the offline solution when the task sequence is long, in terms of both accuracy and stability, which implies the final loss of task t_j (Eqn. (9)) correlates positively with the loss of task t_j after a long sequence of tasks.

6.5 Simulation Time Cost

We show the time cost of the whole standard training (excluding simulation) *vs*. the whole simulation (pseudo-task generation and training) process in GPS in Table 2(a). We take the asynchronous simulation, which applies a binary search to determine $\tilde{s}_j$ sequentially in $O(T \log |\mathcal{M}|)$ sweeps.

[4]In the following text, we refer to "GPS using permutation" as GPS.

Suppose simulating the training of each pseudo-task takes a unit time, each sweep is in $O(T)$. Then, the total simulation process is in $O(T^2 \log |\mathcal{M}|)$, which is dependent on the memory buffer size and the number of simulation training epochs. For efficiency, we train pseudo-tasks with fewer epochs. We disclose those values for each dataset in Appendix D.3. When the task sequence is short, from Table 2(a), we can see the simulation time is over ten times less than the standard training time. When the task sequence is long, we can see the simulation cost grows linearly w.r.t. the number of tasks, as we restrict the search window from $T - j$ to 10 to prevent the simulation cost increasing quadratically.

6.6 Exploration of Other Local Updating Methods

We show the performance of adopting other local updating methods from the existing ER variants besides Eqn. 2, together with GPS. The exemplar base policies we take are from the DER++ and HAL. DER++ leverages knowledge distillation in the episodic memory construction. More specifically, the DER stores the network output logits of random examples in the $\mathcal{M}$ rather than the ground truth labels, while the DER++ stores both. HAL selects pivotal learned data points besides random samples to store in the $\mathcal{M}$.

We change the local updating method from Eqn. (2) to the local updating method used in DER++ or HAL respectively to transplant them into GPS, without any other modifications to the algorithm. From Table 2(b), we can see the performance of DER++ and HAL has been further improved by taking the optimized memory construction for $\mathcal{M}$ by GPS, which implies the ability of GPS as a framework to incorporate advanced memory-based continual learning methods. Besides, as the state-of-the-art method DER++ outperforms other non-ER methods, GPS+DER++ naturally outperforms them.

7 Related Works

Continual Learning Enabling an intelligent agent to learn progressively and adaptively without forgetting old knowledge is a long-standing objective in AI [49, 39]. To combat the catastrophic forgetting problem [30, 18], a few methods have been proposed in continual learning [13, 3, 14]. They usually can be categorized into three classes. One is the regularization-based methods, which introduce an additional regularization term in the loss function to consolidate old behaviors when learning new tasks [20, 44, 53, 9, 19, 36, 16]. One is the replay methods, which store a tiny amount of old examples in a size-bounded memory or condense previous knowledge in a generative model to generate pseudo samples [26, 46, 2, 10, 7, 12, 40, 2, 24, 47, 51, 8, 35]. The data stored in the buffer would be revisited and trained together with each current training task. The third is the parameter isolation methods, which usually allocate additional neural resources for new knowledge without constraints on the model size [42, 29, 45, 52]. The memory cost of these methods would therefore scale with the number of tasks.

The ER Family Experience replay falls into the replay method category, which takes a fixed-sized buffer to store old examples. Existing experience replay variants have adopted the same setup as ER, and focus on refining the local updating method, *i.e.* how to optimally update model parameters by joint training of current task and examples in memory. The advanced techniques used to improve local updating step include additional regularization [10], knowledge distillation [6, 7] and selective memory sampling [2, 35]. Distinct to vanilla ER or its variants, which implicitly minimize the global loss of the final model parameters by designing a powerful local updating method, our work focus on directly optimizing the global objective function by creating pseudo-tasks to mimic the catastrophic forgetting for the current task.

8 Conclusion

In this paper, we propose the dynamic memory construction optimization problem for continual learning under the experience replay setup. The problem aims at finding the best memory construction strategy to optimize the global objective function in continual learning. We simplify the problem to a small space by taking three tactics, and find a solution in time complexity $O(T \log |\mathcal{M}|)$ in an offline setup. We then officially introduce our Global Pseudo-task Simulation (GPS), which provides an approximate solution to the simplified problem under the realistic online setup by creating pseudo-tasks to mimic the future catastrophic forgetting pattern for the current task. Our empirical results have shown our approach outperforms baselines and can improve the accuracy of existing ER variants [10, 7].

Future Studies We focus on the task- and domain-incremental in this work. Some future improvements could be extending Global Pseudo-task Simulation to the class-incremental (class-IL) setup. There are two potential challenges under this setup. 1) the task identity is known. This could be achieved by triggering the simulation after experiencing a certain amount of data points instead of a task. 2) though the ER-Res and ER-Ring-Full mixed policies are good enough under task and domain IL, we might involve a complex mixture policies to find the offline oracle under class-IL, where new assumptions are required.

Further, we hope our work could inspire the community to design new datasets closer to the real-world setting. For example, when task sequences have specific zero-shot transfer patterns. In such cases, we might first infer the zero-shot transfer pattern from a few data points and then inject the bias into the simulation process.

Acknowledgements

We would like to express our sincere gratitude to Sébastien M. R. Arnold, Robin Jia and Jesse Thomason for their valuable suggestions on writing.

References

[1] Learning multiple layers of features from tiny images, 2009.

[2] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval, 2019.

[3] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts, 2017.

[4] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning, 2019.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning.

[6] Ari S. Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space, 2019.

[7] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline, 2020.

[8] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning, 2018.

[9] Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. *Lecture Notes in Computer Science*, page 556–572, 2018.

[10] Arslan Chaudhry, Albert Gordo, Puneet K. Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning, 2021.

[11] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem, 2019.

[12] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning, 2019.

[13] Zhiyuan Chen, Bing Liu, Ronald Brachman, Peter Stone, and Francesca Rossi. *Lifelong Machine Learning*. Morgan amp; Claypool Publishers, 2nd edition, 2018.

[14] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[16] Arthur Douillard, Eduardo Valle, Charles Ollion, Thomas Robert, and Matthieu Cord. Insights from the future for continual learning, 2020.

[17] Jeffrey Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99, 08 1993.

[18] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015.

[19] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks, 2016.

[20] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks, 2017.

[21] Jeremias Knoblauch, Hisham Husain, and Tom Diethe. Optimal continual learning has perfect memory and is np-hard, 2020.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching, 2018.

[25] Sebastian Lee, Sebastian Goldt, and Andrew Saxe. Continual learning in the teacher-student setup: Impact of task similarity, 2021.

[26] Timothée Lesort, Alexander Gepperth, Andrei Stoian, and David Filliat. Marginal replay vs conditional replay for continual learning, 2019.

[27] Zhizhong Li and Derek Hoiem. Learning without forgetting, 2017.

[28] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning, 2017.

[29] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning, 2018.

[30] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989.

[31] Melike Nur Mermer and Mehmet Fatih Amasyali. Training with growing sets: A simple alternative to curriculum learning and self paced learning, 2018.

[32] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning, 2020.

[33] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning?, 2021.

[34] Cuong V. Nguyen, Alessandro Achille, Michael Lam, Tal Hassner, Vijay Mahadevan, and Stefano Soatto. Toward understanding catastrophic forgetting in continual learning, 2019.

[35] Ameya Prabhu, Philip Torr, and Puneet Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *The European Conference on Computer Vision (ECCV)*, August 2020.

[36] Amal Rannen, Rahaf Aljundi, Matthew B. Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[37] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning, 2017.

[38] 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, 2019.

[39] Mark B. Ring. Child: A first step towards continual learning. In *Learning to Learn*, 1998.

[40] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning, 2019.

[41] Sebastian Ruder and Barbara Plank. Learning to select data for transfer learning with bayesian optimization, 2017.

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

[43] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning, 2021.

[44] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning, 2018.

[45] Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task, 2018.

[46] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay, 2017.

[47] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting, 2017.

[48] Stanford. Tiny ImageNet Challenge (CS231n), 2015. `http://tiny-imagenet.herokuapp.com/`.

[49] Sebastian Thrun. Lifelong learning algorithms. In *Learning to Learn*, 1998.

[50] Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. When do curricula work?, 2021.

[51] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning, 2019.

[52] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks, 2017.

[53] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence, 2017.

[54] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See our **Future Studies** of Section 8
   (c) Did you discuss any potential negative societal impacts of your work? [No] No potential negative social impact observed from our perspective
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See our Section 4
   (b) Did you include complete proofs of all theoretical results? [No] The time complexity is self-explanable from the given pseudo algorithms
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] Please refer to our Section 6 and Appendix F to reproduce the experimental results. We will release the github code to the public if our paper gets accepted.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please refer to Appendix F
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Results are averaged over 5 runs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please refer to Appendix F.3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use public data
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No offensive contents observed from our perspectives
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No human subjects/research involved
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

In this appendix, we provide details skipped in the main text. The content is organized as follows:

- Section A. Detailed algorithms of GPS, including binary search and simulation process. (*c.f.* §5.4 of the main text)
- Section B. Validation of the switching point existence. (*c.f.* §4.3 of the main text)
- Section C. Validation on the properties of the simulation methods. (*c.f.* §5.2 of the main text)
- Section D. Additional experimental results and ablation studies. (*c.f.* §6.2, §6.5 and §6.6 of the main text)
- Section E. Exploration of new base policies based on curriculum learning, and their performance with GPS.
- Section F. Experimental details and hyperparameters. (*c.f.* §6.1 of the main text)
- Section G. Algorithms of ER-Res and ER-Ring-Full.

A Detailed Algorithms

A.1 Simulation Process

The pseudocode of our simulation process (Fig. 2 in the main text) is listed in Algorithm 1. We use the notation $P_{1:i}$ to represent the task distributions from P_1 to P_i. Likewise, we use the notation $s_{1:i}$ to represent the switching point from s_1 to s_i. The memory construction function `BuildM` takes as arguments the previous memory and a list of switching points. This function internally checks whether the list of switching points are enough to construct the current memory. If yes, it constructs the memory as described in §4; otherwise, it utilizes the described pseudo-memory construction methods as described in §5.

Algorithm 1 GlobalSim

```
Input: Tested point $a_i$; Pseudo-task distributions $\tilde{P}_{(i+1):T}$; Switching points $s_{1:(i-1)}$; Local updating method $g$; Current model parameters $\theta_i$ and current memory $\mathcal{M}_{i-1}$.
Initialize $\tilde{\theta}_{i:i} \leftarrow \theta_i$
Initialize memory $\tilde{\mathcal{M}}_i \leftarrow$ BuildM($\mathcal{M}_{i-1}, s_{1:(i-1)} \cup \{a_i\}$)
$j \leftarrow i + 1$
while $j \leq T$ do
    Local update: $\tilde{\theta}_{i:j} \leftarrow g(\tilde{\theta}_{i:(j-1)}, \tilde{P}_j, \tilde{\mathcal{M}}_{j-1})$
    Build memory: $\tilde{\mathcal{M}}_j \leftarrow$ BuildM($\tilde{\mathcal{M}}_{j-1}, s_{1:i}$)
    $j \leftarrow j + 1$
end while
Compute loss: $l \leftarrow \mathbb{E}_{(x_j,y_j)\sim P_j} \ell(y_j, f(x_j; \tilde{\theta}_{j:T}))$
return Loss: $l$
```

A.2 Binary Search

In the global binary search as listed in Algorithm 2, we take a search stride $\epsilon = 20$ to increase the robustness of the algorithm.

A.3 GPS Algorithm

Based on the simulation process and binary search, we describe our Global Pseudo-task Simulation method in Algorithm 3.

Algorithm 2 GlobalBS

```
**Input:** Number of tasks $T$; Task distributions $P_{1:i}$; Switching points $s_{1:(i-1)}$; Local updating method $g$; Current model parameters $\theta_i$ and current memory $\mathcal{M}_{i-1}$; Search stride $\epsilon$.
Synthesize pseudo-tasks from $P_i$ with task distributions.
$start \leftarrow 0$
$end \leftarrow |\mathcal{M}|/i$
Loss dictionary: $loss\_dict \leftarrow \varnothing$
**while** $end - start \geq \epsilon$ **do**
    $next \leftarrow (start + end)/2$
    **if** $next$ not in $loss\_dict$ **then**
        $loss \leftarrow$ GlobalSim$(next, \tilde{P}_{(i+1):T}, s_{1:(i-1)}, g, \theta_i, \mathcal{M}_{i-1})$
        $loss\_dict \leftarrow loss\_dict \cup \{next : loss\}$
    **else**
        $loss \leftarrow loss\_dict[next]$
    **end if**
    **if** $next - \epsilon$ not in $loss\_dict$ **then**
        $left\_loss \leftarrow$ GlobalSim$(next - \epsilon, \tilde{P}_{(i+1):T}, s_{1:(i-1)}, g, \theta_i, \mathcal{M}_{i-1})$
        $loss\_dict \leftarrow loss\_dict \cup \{next - \epsilon : left\_loss\}$
    **else**
        $left\_loss \leftarrow loss\_dict[next - \epsilon]$
    **end if**
    **if** $next + \epsilon$ not in $loss\_dict$ **then**
        $right\_loss \leftarrow$ GlobalSim$(next + \epsilon, \tilde{P}_{(i+1):T}, s_{1:(i-1)}, g, \theta_i, \mathcal{M}_{i-1})$
        $loss\_dict \leftarrow loss\_dict \cup \{next + \epsilon : right\_loss\}$
    **else**
        $right\_loss \leftarrow loss\_dict[next + \epsilon]$
    **end if**
    **if** $left\_loss < loss$ **then**
        $end \leftarrow next$
        **continue**
    **else if** $right\_loss < loss$ **then**
        $start \leftarrow next$
        **continue**
    **else**
        $s_i \leftarrow next$
        **break**
    **end if**
**end while**
**if** $s_i$ is not assigned **then**
    $s_i \leftarrow argmin_{loss}(loss\_dict)$
**end if**
**return** Switching point: $s_i$
```

Algorithm 3 Global Pseudo-task Simulation (GPS)

```
**Input:** Number of tasks $T$; Task distributions $P_i, i \in \mathcal{T}$; Local updating method $g(\cdot)$.
Initialize parameters $\theta_0$
Initialize memory $\mathcal{M}_0 = \varnothing$
$i \leftarrow 1$
**while** $i \leq T$ **do**
    Local update: $\theta_i \leftarrow g(\theta_{i-1}, P_i, \mathcal{M}_{i-1})$
    Find switching point: $s_i \leftarrow$ GlobalBS$(T, P_{1:i}, s_{1:(i-1)}, g, \theta_i, \mathcal{M}_{i-1})$
    Build memory: $\mathcal{M}_i \leftarrow$ BuildM$(\mathcal{M}_{i-1}, s_{1:i})$
    $i \leftarrow i + 1$
**end while**
**return** Model parameters: $\theta_T$
```

B Validation of the Switching Point Existence

As a validation of our switching point existence, we further show the switching point of other benchmarks in our experiments. For each benchmark, we plot the global loss sL^G as a function of a_i and select 5 different tasks t_i. Each plot shows clearly the switching point in Fig. 4.

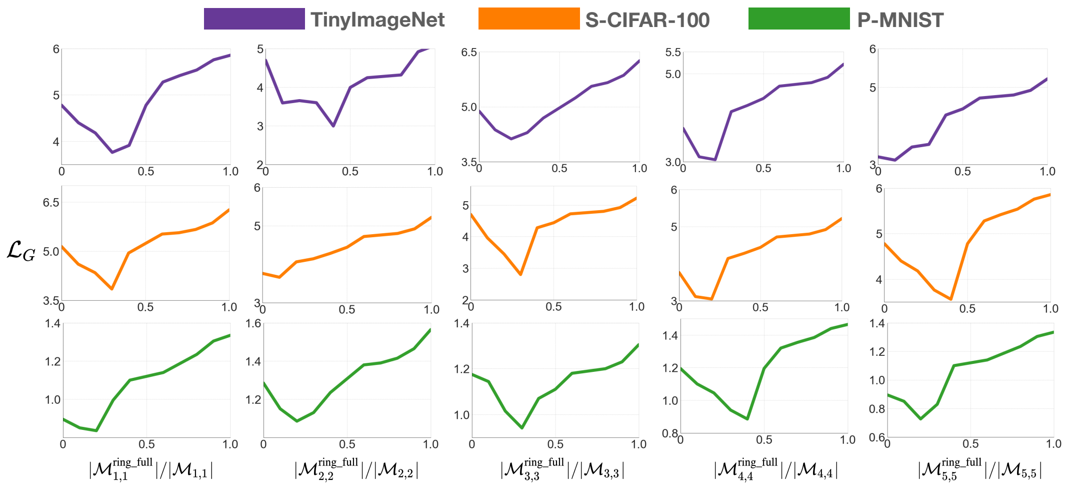


Figure 4: Switching points of the first 5 tasks in three evaluation benchmarks: TinyImageNet, S-CIFAR-100, P-MNIST. We show the change of the global loss, $\mathcal{L}^G$ w.r.t. the ratio of ER-Ring-Full in the memory.

C Validation of the Simulation Method

In this section, we provide the empirical supporting evidences for our hypotheses of the simulation method.

C.1 Task Difficulties

First, we show the task difficulties in the evaluated benchmarks have small variations, as in Table 3. For P-MNIST, S-CIFAR-100 and TinyImageNet, we evaluate the first 5 tasks end-to-end for simplicity. For S-CIFAR-10, we evaluate all the tasks end-to-end. Further, we evaluate the difficulty along the

Table 3: Accuracy and variance of accuracy of tasks from four vision benchmarks trained end-to-end.

Dataset	Task 1	Task 2	Task 3	Task 4	Task 5	Variance
P-MNIST	97.48	97.28	97.33	97.78	97.53	0.03
S-CIFAR-10	98.20	94.85	96.50	98.90	98.15	2.18
S-CIFAR-100	85.70	87.70	88.10	88.10	86.20	1.02
TinyImageNet	78.20	76.80	77.30	76.50	77.20	0.33

pseudo-task sequence synthesized from the first task of S-CIFAR-100 by permutation, rotation and blurring, we generate 5 tasks for each simulation method. The results in Table 4 shows permutation and rotation generate tasks with similar difficulties as the original S-CIFAR-100 tasks, while blurring generate tasks with increasing difficulties as the task sequence grows.

C.2 Forward Transfer Ability of Tasks

Next, we explore the forward transfer ability of the pseudo-tasks *vs*. the real tasks. We evaluate the model trained on the first two tasks of S-CIFAR-100 on task sequences different by different simulation methods.

Table 4: Accuracy and variance of accuracy of pseudo-tasks synthesized by different methods from the first task of S-CIFAR-100. The Task i column refers to the end-to-end training of the pseudo-task i.

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Variance
Permutation	85.50	85.70	85.30	86.10	85.10	0.11
Rotation	85.40	85.40	85.70	84.90	84.50	0.18
Blurring	83.90	81.70	78.90	74.40	69.50	26.80

Table 5 shows that the permutation pseudo-tasks and the real tasks both allows zero transfer ability, as the random guess accuracy of a 10-class classification is 10%. Rotation, instead, creates a task sequence that allows nearly perfect forward transfer ability. Blurring creates a task sequence which allows some forward transfer ability from the beginning, but it gradually reduces to a random guess as task difficulty grows.

Table 5: Forward transfer ability of different simulation methods after training task t_1 and task t_2 on S-CIFAR-100. Numbers are the accuracy of the task.

Dataset	Method	Task 3	Task 4	Task 5	Task 6	Task 7
S-CIFAR-100	Real	11.80	10.40	10.30	9.40	9.70
	Permutation	10.50	10.40	10.30	10.10	11.10
	Rotation	75.40	78.10	77.70	79.90	80.50
	Blurring	70.90	65.70	52.90	30.40	15.50

D Additional Experimental Results

D.1 Accuracy of GPS with Small Memory Buffer

We have also carried out experiments of GPS to evaluate its performance when the memory buffer is relatively small, as shown in Table 6. With a small size memory buffer, GPS does not show significant improvement. One reason is the switching point s_i is very close to 1, *i.e.*, taking the pure ER-Ring-Full policy is good enough. Another cause is our base policy assumption does not work very well under the small memory buffer size as model becomes more sensitive to points selection under small $\mathcal{M}$.

Table 6: Accuracy of GPS using permutation and ER baselines on four datasets with smaller buffer sizes.

Method	**P-MNIST**	**S-CIFAR-10**	**S-CIFAR-100**	**TinyImageNet**
$\vert\mathcal{M}\vert$	100	20	200	200
ER-Res	$65.59_{\pm 1.38}$	$80.68_{\pm 2.28}$	$64.99_{\pm 1.74}$	$38.60_{\pm 0.74}$
ER-Ring-Full	$66.10_{\pm 1.36}$	$81.30_{\pm 1.98}$	$65.95_{\pm 0.96}$	$40.85_{\pm 0.78}$
ER-Hybrid	$66.30_{\pm 1.21}$	$81.43_{\pm 2.54}$	$66.30_{\pm 1.32}$	$40.75_{\pm 0.54}$
GPS	$\mathbf{66.50}_{\pm 1.11}$	$\mathbf{81.78}_{\pm 1.56}$	$\mathbf{66.51}_{\pm 0.88}$	$\mathbf{40.89}_{\pm 0.45}$

D.2 Results of GPS with DER++, HAL and Baselines

We put complete results of GPS+DER, GPS+DER++, GPS+HAL and baselines in Table 7. Note the A-GEM [11], iCaRL [37] and GSS [4] use the same memory size as the ER series for fair comparison. The results stand as a complete empirical support to illustrate that the performance of other ER variants have been improved after using our GPS method. We also reported the results of GPS+DER, GPS+DER++ and GPS+HAL with smaller memory sizes in Table 8.

Table 7: Accuracy of GPS using permutation incorporating DER, DER++ [7] and HAL [10], comparing to other methods. '-' indicates experiments we were unable to run, due to compatibility issues (e.g. Domain IL for iCaRL) or intractable training time or memory utilization (e.g. OGD, GSS on TinyImageNet).

	P-MNIST	**S-CIFAR10**	**S-CIFAR100**	**TinyImageNet**
oEWC	$69.21_{\pm 2.92}$	$62.97_{\pm 3.55}$	$55.37_{\pm 2.71}$	$20.81_{\pm 0.95}$
iCaRL	-	$88.97_{\pm 2.77}$	$78.21_{\pm 1.01}$	$38.77_{\pm 3.68}$
GSS	$86.34_{\pm 4.28}$	$87.80_{\pm 2.71}$	$77.34_{\pm 3.21}$	-
A-GEM	$77.36_{\pm 1.28}$	$83.87_{\pm 1.55}$	$69.61_{\pm 1.47}$	$25.30_{\pm 0.87}$
OGD	$81.52_{\pm 2.21}$	-	-	-
	P-MNIST	**S-CIFAR10**	**S-CIFAR100**	**TinyImageNet**
$\lvert\mathcal{M}\rvert$	1000	200	2000	2000
HAL	$87.69_{\pm 0.34}$	$89.29_{\pm 1.31}$	$80.81_{\pm 1.21}$	$61.27_{\pm 1.10}$
GPS+HAL	$\mathbf{88.73}_{\pm \mathbf{0.03}}$	$\mathbf{91.27}_{\pm \mathbf{0.93}}$	$\mathbf{82.33}_{\pm \mathbf{0.47}}$	$\mathbf{63.24}_{\pm \mathbf{0.80}}$
DER	$90.47_{\pm 2.69}$	$91.04_{\pm 0.18}$	$81.78_{\pm 0.50}$	$60.90_{\pm 1.08}$
GPS+DER	$\mathbf{90.27}_{\pm \mathbf{1.78}}$	$\mathbf{91.53}_{\pm \mathbf{0.13}}$	$\mathbf{83.39}_{\pm \mathbf{0.44}}$	$\mathbf{61.89}_{\pm \mathbf{1.06}}$
DER++	$91.14_{\pm 0.22}$	$92.06_{\pm 0.20}$	$82.20_{\pm 0.89}$	$62.67_{\pm 1.08}$
GPS+DER++	$\mathbf{91.84}_{\pm \mathbf{0.16}}$	$\mathbf{92.57}_{\pm \mathbf{0.10}}$	$\mathbf{83.53}_{\pm \mathbf{0.64}}$	$\mathbf{63.01}_{\pm \mathbf{0.98}}$

Table 8: Accuracy of GPS using permutation incorporating DER, DER++ [7] and HAL [10] with small memory sizes $|\mathcal{M}|$.

	P-MNIST	**S-CIFAR10**	**S-CIFAR100**	**TinyImageNet**
$\lvert\mathcal{M}\rvert$	100	20	200	200
HAL	$80.77_{\pm 1.31}$	$82.56_{\pm 2.01}$	$52.89_{\pm 0.97}$	$38.64_{\pm 0.89}$
GPS+HAL	$\mathbf{81.45}_{\pm \mathbf{0.94}}$	$\mathbf{83.74}_{\pm \mathbf{1.75}}$	$\mathbf{53.57}_{\pm \mathbf{0.77}}$	$\mathbf{39.37}_{\pm \mathbf{0.44}}$
DER	$81.72_{\pm 1.11}$	$85.57_{\pm 1.59}$	$57.51_{\pm 0.60}$	$40.21_{\pm 0.77}$
GPS+DER	$\mathbf{82.04}_{\pm \mathbf{0.97}}$	$\mathbf{85.68}_{\pm \mathbf{1.49}}$	$\mathbf{57.83}_{\pm \mathbf{0.59}}$	$\mathbf{40.54}_{\pm \mathbf{0.54}}$
DER++	$83.57_{\pm 0.59}$	$83.45_{\pm 1.76}$	$\mathbf{58.18}_{\pm \mathbf{1.08}}$	$40.67_{\pm 1.16}$
GPS+DER++	$\mathbf{83.86}_{\pm \mathbf{0.35}}$	$\mathbf{83.57}_{\pm \mathbf{1.45}}$	$58.15_{\pm 0.78}$	$\mathbf{40.70}_{\pm \mathbf{1.03}}$

E New Base Policies based on Curriculum Learning

To further test the power of GPS, we substitute the base policies with two novel memory construction methods designed by us based on curriculum learning [17], ER-CurRes and ER-CurRing-Full. The inspiration of these two methods comes from recent findings that curriculum can help when noisy data are present [50, 31, 41]. We believe data points of future task can be viewed as noisy interference for samples stored in $\mathcal{M}$. In these two policies, curricular easy points of each task are picked as candidates for $\mathcal{M}$.

E.1 Algorithm

Curricular Easy Samples We rank data examples from easy to hard based on the implicit curricula [50]. Specifically, we first record the learned epochs as an attribute of an example, which is the earliest epoch in training where a model correctly predicts this example for that and subsequent epochs till now. As the learned epoch is a positive integer attribute, it is defined as a subset of the totally ordered set $\mathbb{Z}^+$. We also record the current loss of each example as another attribute. The loss attribute is defined as a subset of the totally ordered set $\mathbb{R}$. The ranking of examples is based on the lexicographical order on the Cartesian product of the two attributes, *i.e.*, first sorting the examples by the learned epoch attribute, and then ordering examples within the same ranked epoch by their losses. As a result, each training example of a task would be associated with an unique ranking. Also, the ranking would be updated after each epoch.

Algorithm 4 ER-CurRes (for a single task)

```
Input: Reservoir memory buffer $\mathcal{M}$; Number of epochs $k$; Task distribution $P$; Dataset size $|\mathcal{D}|$;
Batch size $B$; Portion of easy data $\gamma$; Model parameters $\theta$; Seen examples $N$.
Initialize a random easy pool $P^{\text{easy}}$ from $P$
for $ep \in \{1, ..., k\}$ do
    for $iter \in 1, ..., |\mathcal{D}|/B$ do
        Sample a batch $B_P$ from $P$ and a batch $B_{\mathcal{M}}$ from $\mathcal{M}$
        Update $\theta$ with $B_P \cup B_{\mathcal{M}}$
        if $ep \leq \lceil k/2 \rceil$ then
            Update $\mathcal{M}$ with a probability $|\mathcal{M}|/N$ for each examples in $B_P$
            $N = N + 1$
        else
            Update $\mathcal{M}$ with a probability $|\mathcal{M}|/(\gamma * N)$ for each examples in $B_P \cap P^{\text{easy}}$
            $N = N + 1/\gamma$
        end if
    end for
    Update $P^{\text{easy}}$: order examples based on the implicit curriculum and select the first $\gamma|\mathcal{D}|$
end for
Select the memory for the current task as $\mathcal{M}_{\text{now}}$ and the memory for all previous tasks as $\mathcal{M}_{\text{past}}$
for $idx \in \mathcal{M}_{\text{now}}$ do
    if $idx \notin P^{\text{easy}}$ then
        Replace the slot in $\mathcal{M}_{\text{now}}$ with samples from ($P^{\text{easy}} - \mathcal{M}_{\text{now}}$)
    end if
end for
Return Updated $\theta$ and $\mathcal{M} = \mathcal{M}_{\text{now}} \cup \mathcal{M}_{\text{past}}$
```

ER-CurRes The ER-CurRes algorithm is shown in Algorithm. 4. Different from ER-Res which stores examples sampled from the whole distribution P_i of a task t_i [12], we sample subset of data points from an "easy pool", *i.e.* P_i^{easy}. The size is calculated by the dataset size $|\mathcal{D}_i|$ multiplying a hyperparameter γ, whose value is reported for each evaluation dataset in Section F. Compared to taking the top few easiest points of each class, sampling from the pool utilizes the benefits of randomness [5, 50]. Suppose we train a total of k epochs of task t_i, in order to obtain a smooth transition and the samples based on a more stable curriculum ranking, we take the examples obeying ER-Res policy (*i.e.*, samples from P_i) for the first $\lceil k/2 \rceil$ epochs, and take the examples obeying ER-CurRes policy (*i.e.*, samples from P_i^{easy}) for the last $\lfloor k/2 \rfloor$ epochs. When we finish training, we replace the examples of task t_i in the memory which are not from P_i^{easy}.

ER-CurRing-Full Likewise, as shown in Algorithm. 5, ER-CurRing-Full follows the ER-Ring-Full strategy in the first $\lceil k/2 \rceil$ epochs to fill a FIFO memory [12]. After that, we substitute the memory slots with points from the easy pool of each observed class. In the construction of the easy pool, instead of taking the easiest $\gamma|\mathcal{D}|$ examples as in ER-CurRes, we use the easiest $\gamma|\mathcal{D}|/C$ examples of each class based on the ranking, where C is the number of classes.

E.2 Experimental Results of GPS w/ Cur

The accuracy comparison between GPS w/ Cur and GPS w/ ER-CurRes, ER-CurRing-Full are shown in Table 9. From the table, we can see GPS w/ Cur outperforms both ER-CurRes and ER-CurRing-Full in both datasets. Moreover, it shows leveraging curriculum in base policies of GPS can further improve its performance compared to its plain version.

F Experimental Details

F.1 Simulation Details

In experiments, we set the number of examples in each synthesized pseudo-task the same as the size of the memory buffer, *i.e.*, if $|\mathcal{M}| = 1000$, we then generate 1000 examples for each pseudo-task. For computational efficiency, we set the number of training epochs small in the simulated training process.

Algorithm 5 ER-CurRing-Full (for a single task)

```
**Input:** Ring-Full memory buffer $\mathcal{M}$; Number of epochs $k$; Task distribution $P$; Dataset size $|\mathcal{D}|$; Batch size $B$; Portion of easy data $\gamma$; Model parameters $\theta$.
Initialize a random easy pool $P^{\text{easy}}$ from $P$
Reallocate the memory $\mathcal{M}_{\text{past}}$ for all previous tasks, and allocate the memory $\mathcal{M}_{\text{now}}$ for the current task
for $e \in \{1, ..., k\}$ do
    for $iter \in 1, ..., |\mathcal{D}|/B$ do
        Sample a batch $B_P$ from $P$ and a batch $B_{\mathcal{M}}$ from $\mathcal{M}$
        Update $\theta$ with $B_P \cup B_{\mathcal{M}}$
        if $e \leq \lceil k/2 \rceil$ then
            Update $\mathcal{M}_{\text{now}}$ with $B_P$
        else
            Update $\mathcal{M}_{\text{now}}$ with $B_P \cap P^{\text{easy}}$
        end if
    end forUpdate $P^{\text{easy}}$: order examples based on the implicit curriculum and select the first $\gamma$ portion of each class
end for
for $idx \in \mathcal{M}_{\text{now}}$ do
    if $idx \notin P^{\text{easy}}$ then
        Replace the slot in $\mathcal{M}_{\text{now}}$ with samples from ($P^{\text{easy}} - \mathcal{M}_{\text{now}}$)
    end if
end for
**Return** Updated $\theta$ and $\mathcal{M} = \mathcal{M}_{\text{now}} \cup \mathcal{M}_{\text{past}}$
```

Table 9: Accuracy of GPS using curriculum-based policies *vs*. the corresponding baselines.

	P-MNIST	**S-CIFAR10**	**S-CIFAR100**	**TinyImageNet**
$\lvert\mathcal{M}\rvert$	1000	200	2000	2000
ER-CurRes	$86.78_{\pm 0.49}$	$92.47_{\pm 0.20}$	$81.38_{\pm 0.51}$	$61.89_{\pm 0.03}$
ER-CurRing-Full	$86.16_{\pm 0.49}$	$91.70_{\pm 0.50}$	$81.16_{\pm 0.65}$	$61.03_{\pm 0.42}$
GPS w/ Cur	$\mathbf{87.35}_{\pm 0.18}$	$\mathbf{93.08}_{\pm 0.17}$	$\mathbf{82.34}_{\pm 0.79}$	$\mathbf{62.88}_{\pm 0.03}$
	P-MNIST	**S-CIFAR10**	**S-CIFAR100**	**TinyImageNet**
$\lvert\mathcal{M}\rvert$	100	20	200	200
ER-CurRes	$65.34_{\pm 0.69}$	$81.59_{\pm 3.23}$	$65.43_{\pm 1.37}$	$40.75_{\pm 0.98}$
ER-CurRing-Full	$66.42_{\pm 1.03}$	$81.54_{\pm 2.46}$	$66.73_{\pm 0.12}$	$41.54_{\pm 0.57}$
GPS w/ Cur	$\mathbf{66.92}_{\pm 0.54}$	$\mathbf{81.68}_{\pm 1.98}$	$\mathbf{67.08}_{\pm 0.36}$	$\mathbf{41.80}_{\pm 0.49}$

We train 1 epoch for pseudo-tasks synthesized in the P-MNIST dataset, 3 epochs for pseudo-tasks in S-CIFAR-10, S-CIFAR-100 and TinyImageNet. As for the batch size, the optimizer and the learning step during the simulation process, they are all the same as in the real training process.

F.2 Other Hyperparameters

We disclose the experimental hyperparameters values not reported in the main manuscript in Table 10. In the table, γ in the 'Cur-' series methods is the easy pool ratio of the curriculum-based policies as we discussed in Section E, while other symbols refer to the respective methods. In all the experimental evaluation by accuracy, reported numbers are averaged over 5 runs.

F.3 Time Measurement

We measure our training and simulation time for each dataset in a single NVIDIA Tesla K80 GPU for fair comparison. The time we report is the total processing time averaged on 5 runs, assessed in wall-clock time (seconds) at the end of the last task and then converted into minutes.

Table 10: Other hyperparameters used in our experiments.

Dataset	Method	Parameters
P-MNIST	CurER-Res	γ: 0.2
	CurER-Ring-Full	γ: 0.1
	GPS w/ Cur	γ: 0.2
	DER	α: 0.5
	DER++	α: 1.0 β: 0.5
	GPS+DER	α: 0.5
	GPS+DER++	α: 1.0 β: 0.5
	HAL	λ: 0.1 β: 0.5 γ: 0.1
	GPS+HAL	λ: 0.1 β: 0.5 γ: 0.1
	oEWC	λ: 0.7 γ: 1.0
	GSS	$gmbs$: 10 nb: 1
	OGD	stored gradients : 100/task (perm)
S-CIFAR-10	CurER-Res	γ: 0.2
	CurER-Ring-Full	γ: 0.1
	GPS w/ Cur	γ: 0.2
	DER	α: 0.3
	DER++	α: 0.1 β: 0.5
	GPS+DER	α: 0.3
	GPS+DER++	α: 0.1 β: 0.5
	HAL	λ: 0.1 β: 0.5 γ: 0.1
	GPS+HAL	λ: 0.1 β: 0.5 γ: 0.1
	oEWC	λ: 0.7 γ: 1.0
	iCaRL	wd: 0
	GSS	$gmbs$: 32 nb: 1
S-CIFAR-100	CurER-Res	γ: 0.2
	CurER-Ring-Full	γ: 0.1
	GPS w/ Cur	γ: 0.2
	DER	α: 0.5
	DER++	α: 0.5 β: 0.5
	GPS+DER	α: 0.5
	GPS+DER++	α: 0.5 β: 0.5
	HAL	λ: 0.1 β: 0.5 γ: 0.1
	GPS+HAL	λ: 0.1 β: 0.5 γ: 0.1
	oEWC	λ: 0.7 γ: 1.0
	iCaRL	wd: 10^{-5}
	GSS	$gmbs$: 32 nb: 1
TinyImageNet	CurER-Res	γ: 0.2
	GPS w/ Cur	γ: 0.2
	CurER-Ring-Full	γ: 0.1
	DER	α: 0.1
	DER++	α: 0.1 β: 0.5
	GPS+DER	α: 0.1
	GPS+DER++	α: 0.1 β: 0.5
	HAL	λ: 0.1 β: 0.5 γ: 0.1
	GPS+HAL	λ: 0.1 β: 0.5 γ: 0.1
	oEWC	λ: 0.7 γ: 1.0
	iCaRL	wd: 10^{-5}

G ER-Res & ER-Ring-Full Algorithms

We put the algorithms of ER-Res [12] and ER-Ring-Full [12] for a single task in Alg. 6 and Alg. 7 for reference.

Algorithm 6 ER-Res. $|\mathcal{M}|$ is the number of examples the memory can store, t is the task id, n is the number of examples observed so far in the data stream, and B is the input mini-batch.

```
1: procedure UPDATEMEMORY($|\mathcal{M}|, t, n, B$)
2:     $j \leftarrow 0$
3:     for $(x, y)$ in $B$ do
4:         $M \leftarrow \mathcal{M}$
5:         if $M < |\mathcal{M}|$ then
6:             $\mathcal{M}$.append$(x, y, t)$
7:         else
8:             $i = randint(0, n + j)$
9:             if $i < |\mathcal{M}|$ then
10:            end if
11:        end if
12:        $j \leftarrow j + 1$
13:    end for
14:    return $\mathcal{M}$
15: end procedure
```

Algorithm 7 ER-Ring-Full.

```
1: procedure UPDATEMEMORY($|\mathcal{M}|, t, n, B$)
2:     for $(x, y)$ in $B$ do
3:         Divide memory into FIFO stacks $\mathcal{M}[y]$, where $|\mathcal{M}[y]| = |\mathcal{M}|/t$
4:         $\mathcal{M}[y]$.append$(x)$
5:     end for
6:     return $\mathcal{M}$
7: end procedure
```