Improving Replay Sample Selection and Storage for Less Forgetting in Continual Learning

Daniel Brignac
University of Arizona
Tucson, Arizona
dbrignac@arizona.edu

Niels Lobo
University of Central Florida
Orlando, Florida
niels@cs.ucf.edu

Abhijit Mahalanobis
University of Arizona
Tucson, Arizona
amahalan@arizona.edu

Abstract

Continual learning seeks to enable deep learners to train on a series of tasks of unknown length without suffering from the catastrophic forgetting of previous tasks. One effective solution is replay, which involves storing few previous experiences in memory and replaying them when learning the current task. However, there is still room for improvement when it comes to selecting the most informative samples for storage and determining the optimal number of samples to be stored. This study aims to address these issues with a novel comparison of the commonly used reservoir sampling to various alternative population strategies and providing a novel detailed analysis of how to find the optimal number of stored samples.

1. Introduction

Deep learning has revolutionized the field of computer vision, achieving human-like capabilities of image understanding and perception. Unlike humans, however, deep learners consistently struggle to adopt new knowledge while maintaining performance on previously learned tasks. This is the problem known as catastrophic forgetting [25] in which a learner's performance significantly diminishes as it acquires knowledge for new tasks. This motivates the study of continual learning [30, 38] to address the problem of catastrophic forgetting.

In continual learning, a learner is presented with a sequence of tasks of unknown length where the only data available is that of the current task at hand. As new tasks arrive, we wish to learn each new task while preserving the knowledge learned from previous tasks. During inference, we may be presented with data from any of the previously learned tasks, thus the retention of previous knowledge is imperative when adapting to new tasks.

The inference stage primarily comes in three flavors: task-incremental learning (task-IL), domain-incremental learning (domain-IL) and class-incremental learning (class-IL) [39]. Task-IL and domain-IL are generally considered the easier scenarios as we are either given the task-ID at test time in task-IL, or we must only solve for the current task at hand in domain-IL. Class-IL is significantly more challenging as we must infer for all tasks seen so far without the revealing of a task-ID. Because of this, class-IL has become a primary focus of recent continual learning works [3, 27, 5, 36].

Replay [29, 32, 4] is a commonly used approach to remedy the problem of catastrophic forgetting in both class-IL and task-IL. The concept of replay draws inspiration from Complementary Learning Systems theory of humans which posits that recent experiences stored in the hippocampus develop connections to the neocortex that become ingrained over time to eventually be encoded in long-term memory [24, 13]. As such, when replay is employed in artificial learners, a small amount of previously learned data is stored in memory to then be "replayed" during the training of the current task to emphasize the previously encoded connections of the learner and thus avoid forgetting.

When using replay methods, it is important to have a small canonical set of exemplars stored in memory that effectively capture the underlying class distributions of the dataset. Thus, the selection of *which samples to store* is a non-trivial task. Previous replay methods rely on reservoir sampling [32, 4, 1, 5, 2] to populate memory with past data. As reservoir sampling is a random sampling method, this could lead to the storage of redundant and potentially insignificantly informative data points in memory causing replay methods to not perform to their maximal capability as recently demonstrated in [37, 43].

There is also no consensus regarding *how many samples should be stored* to be used for replay. Naturally, as we store more samples in memory, we expect performance to increase, however, as this number grows, we start to deviate from the constraints of continual learning where we seek to store minimal information. There must exist some small optimal number of samples to be stored to make maximum

use of replay methods.

In this work we study the two questions *which samples to store*, and *how many samples should be stored* in memory by comparing the commonly used approach of reservoir sampling to three other memory population strategies. We show through extensive empirical evaluation that reservoir sampling leads to greater forgetting when compared with more strategic population approaches. We additionally detail two methods to address the question of how many samples should be stored based on an analysis of significant eigenvectors and eigenvalues. We proceed to show that memory populated according to these two criteria leads to overall more competitiveness and better performance of all population strategies.

2. Related Works

Continual learning methods can generally be grouped into three categories: (i) regularization methods, (ii) architectural methods and (iii) replay methods. Regularization methods introduce a new loss term meant to penalize significant drift from previously learned parameters [19, 22, 17, 41]. Architectural methods seek to adapt existing network architectures such that various portions of the model contain global and/or shared knowledge while others contain task-specific knowledge [26, 44, 35, 33, 27]. Historically, regularization methods and architectural methods underperform when compared directly to replay methods further motivating the study of and improvements to replay.

In replay methods, we are allowed to store some small subset of data in a memory buffer and selectively "replay" samples from this buffer when training on the current task to maintain previous task performance. Earlier works include [23] and [10], both of which use the memory buffer as a constraint on gradient updates to ensure that loss remains low on the buffer samples. These gradient-based methods for replay in general show poor performance when compared to experience replay methods [29, 32, 4, 1, 9, 7, 5]. In experience replay, it is common practice to store any of the raw data sample, the label, the logits, or a combination of all three. All the the aforementioned experience replay methods select the samples to be stored by some random sampling method, such as reservoir sampling or uniform sampling, leading to the potential storage of insignificant data.

A number of previous works suggest to either populate the memory buffer with some fixed arbitrary number of samples as new classes are encountered or to empirically find optimal buffer size for a desired task performance [1, 31]. These methods for expanding the buffer are rooted in heuristics and thus not necessarily optimal further motivating the study for optimal growth of the buffer for learning new tasks.

3. Methodology

Algorithm 1 Memory Buffer Population

Input: $\mathcal{T}$, $\mathcal{D}_t$, $\mathcal{M}$, buffer size (if not dynamic), populate $\in$ {*reservoir, herding, GSS, IPM*}, training-strategy

$\mathcal{M} \leftarrow \emptyset$
for t in $\mathcal{T}$ **do**
  for *minibatch* in $\mathcal{D}_t$ **do**
    if $\mathcal{M} = \emptyset$ **then**
      train($\mathcal{D}_t$) according to training-strategy
      if populate $\in$ {*resrvoir, GSS*}, **then**
        $\mathcal{M} \leftarrow$ populate($minibatch$, buffer size)
    else
      train($\mathcal{D}_t \cup \mathcal{M}$) according to training-strategy
      if populate $\in$ {*resrvoir, GSS*}, **then**
        $\mathcal{M} \leftarrow$ populate($minibatch$, buffer size)
  if populate $\in$ {*herding, IPM*}, **then**
    $\mathcal{M} \leftarrow$ populate($\mathcal{D}_t$, buffer size)

Consider a sequence of $\mathcal{T}$ tasks where each task has an associated dataset $\mathcal{D}_t = \{(x_i^t, y_i^t)\}$ for each $t \in \{1, 2, ..., \mathcal{T}\}$ and $i = 1, ..., N_t$ where x_i^t denotes the i^{th} sample of the t^{th} task, y_i^t its associated ground truth label, and N_t the number of samples in task t. We assume each task t contains a unique, non-overlapping set of classes drawn from an i.i.d distribution. In our continual learning formulation, we seek to sequentially learn each $\mathcal{D}_t$ in an offline setting while maintaining performance on every $\mathcal{D}_k$ for $k < t$ by employing a memory buffer $\mathcal{M}$ which is populated according to a specific population strategy. For $t = 1$, we train only on $\mathcal{D}_t$ and for $t > 1$, we train on $\mathcal{D}_t \cup \mathcal{M}$.

Since we are primarily concerned with strategic buffer population, the training method in which we perform continual learning is agnostic of the memory buffer population. Thus, we can use any training strategy along with any buffer population strategy studied herein. We give an overview of each population strategy studied and then present a novel scheme to identify how many samples per class should be stored in $\mathcal{M}$.

We make a note of the distinction between fixed memory buffers and dynamic memory buffers. A fixed memory buffer refers to a buffer $\mathcal{M}$ that is fixed in the amount of data it can hold (*e.g.*, a fixed buffer size of 200 can contain a maximum of 200 data samples) while a dynamic buffer may grow in size as new classes are encountered. We denote the size of the fixed buffer as $|\mathcal{M}|$. Traditionally, fixed memory buffers are used whenever replay methods are employed. To the best of our knowledge, we are the first to consider dynamic memory buffers as discussed in Section 3.5. When the buffer is not described to be either fixed or dynamic, the method is agnostic of buffer type. An overview of our

described approach is given in Algorithm 1.

3.1. Reservoir Sampling

In reservoir sampling [40], we are presented with a stream of data in which we randomly sample from the stream and store each sample in $\mathcal{M}$. If a sample is selected to be stored when $\mathcal{M}$ is saturated, we then randomly replace a sample in $\mathcal{M}$ with the current selected sample to be stored.

In practice, reservoir sampling tends to favor the storage of samples from earlier encountered tasks. This causes the minimal storage of samples from downstream tasks and an overall unbalanced fixed buffer leading to greater forgetting, particularly when the fixed buffer size is small.

In addition to favoring the storage of earlier encountered task data, reservoir sampling has no mechanism to differentiate between whether a selected sample to be stored is informative or redundant. This leads to the potential storage of insignificant data which in turn can diminish the network's previously learned decision boundaries [3]. This motivates the study of memory buffer population strategies that can always store the next most informative sample and mitigate forgetting.

3.2. The Herding Algorithm

A natural first choice of substitute to reservoir sampling is the herding algorithm as proposed in iCaRL [29] which is an extension of Welling's herding in [42]. Herding seeks to store samples that best represent the sample's class mean in feature space. In this sense, the herding algorithm can be thought of as a greedy mean preserving scheme as each selected sample is the closest to its learned class mean.

Formally, for a learned class mean μ_c in D-dimensional feature space and a mapping from image space to feature space given by $\phi : x_i^{t,c} \rightarrow \mathbb{R}^D$ where c denotes the class label of image x_i^t, herding seeks to find the sample $x_i^{t,c}$ to add to the class exemplar set P_c in memory buffer $\mathcal{M}$ that minimizes the distance to μ_c as

$$\underset{x_i^{t,c}}{\operatorname{argmin}} \left|\left| \mu_c - \frac{1}{K}\left(\phi(x_i^{t,c}) + \sum_{j=1}^{K-1}\phi(p_j)\right)\right|\right|_2 \quad (1)$$

where K denotes the number of samples to be stored for class c and p_j denotes a sample j belonging to P_c.

Because herding relies on the well learned features for each class to store samples, herding should be performed at the conclusion of each task. This makes herding most suitable for the offline continual learning scenario in which we are allowed multiple iterations through $\mathcal{D}_t$ before moving to the next subsequent task.

3.3. Gradient Sample Selection

Gradient sample selection (GSS) is taken from [2] where the objective is to optimize the loss on current task data while maintaining minimal loss on previous task data. To ensure that loss remains low on previous tasks, GSS seeks to maximize the diversity of the buffer $\mathcal{M}$ by selecting samples whose gradient angle is maximal compared to all other samples currently stored (*i.e.*, maximizing gradient direction variance). This formulation involves solving a quadratic integer programming of polynomial complexity w.r.t. the buffer size thus, a greedy formulation of GSS is used instead.

In the greedy method, GSS maintains a score r_i for each sample i in the buffer based on the maximal cosine similarity of the current sample with the other samples in the buffer given by

$$r_i = \max_i \frac{\langle g_i, G\rangle}{||g_i||_2||G||_2} \quad (2)$$

where g_i and G denote the gradients of the current sample i and the set of samples stored in $\mathcal{M}$ respectively.

While GSS aims to select and store samples based on maintaining maximum variance of gradient direction, there is still a randomness component in determining samples to potentially be replaced. When a sample is selected and deemed appropriate for replacement, there is no guarantee that this is a least informative sample, as determined by GSS, and could thus potentially lead to greater forgetting. In addition, GSS has no mechanism for class balancing as samples are randomly added and replaced which can further hinder performance.

3.4. Iterative Projection and Matching

Iterative projection and matching (IPM) [16] is drawn from active learning in which we seek to find the most informative data points for training. Here, we adopt IPM to select the most informative data points for storage in $\mathcal{M}$ to address the shortcomings with reservoir sampling while also maintaining a balanced buffer similar to the strategy used in herding.

Let $\boldsymbol{A}_c \in \mathbb{R}^{N_c \times D}$ denote the matrix of features $\phi(x_1^t), ..., \phi(x_{N_c}^t) \in \mathbb{R}^D$ where N_c denotes the number of samples in class c, D the dimension of features, and $\phi : \mathbb{R}^{H\times W\times C} \rightarrow \mathbb{R}^D$ a mapping from image space to feature space (*i.e.*, the learned network). The n^{th} row in $\boldsymbol{A}_c$ is formed by $\phi(x_{n_c}^t)^T$. We seek to reduce $\boldsymbol{A}_c$ to some $\boldsymbol{A}_R \in \mathbb{R}^{K\times D}$ where K is the number of samples to be stored from class c.

Let $T \subset \{1, ..., N_c\}$ with $|T| = K$ be the set of selected samples. Then, we can project the rows of $\boldsymbol{A}_c$ onto the span of the selected rows indexed by T. We denote this operation as $\boldsymbol{\omega}_T(\boldsymbol{A}_c)$. As done in [11], we can now cast the replay

sample selection problem as the optimization problem

$$\underset{|T|=K}{\operatorname{argmin}} ||\boldsymbol{A}_c - \boldsymbol{\omega}_T(\boldsymbol{A}_c)||_F^2 \tag{3}$$

where $||\cdot||_F$ is the Frobenius norm. This problem is NP-Hard however as we must search all subsets T over $\boldsymbol{A}_c$ [46]. Thus, we use the IPM [16] algorithm to approximate 3.

We can express $\boldsymbol{\omega}_T(\boldsymbol{A}_c)$ as a rank-K factorization $\boldsymbol{U}\boldsymbol{V}^T$ where $\boldsymbol{U} \in \mathbb{R}^{N_c \times K}$ and $\boldsymbol{V}^T \in \mathbb{R}^{K \times D}$ and modify 3 by recasting it as two sub-problems [16, 15]

$$(u, v) = \underset{u,v}{\operatorname{argmin}} ||\boldsymbol{A}_c - \boldsymbol{u}\boldsymbol{v}^T||_F^2 \text{ s.t. } ||\boldsymbol{v}|| = 1 \tag{4}$$

$$m^{(1)} = \underset{m}{\operatorname{argmax}} |\boldsymbol{v}^T \boldsymbol{\rho}| \tag{5}$$

where $\boldsymbol{\rho} = \phi(x_{n_c}^t)/||\phi(x_{n_c}^t)||_2$, $m^{(1)}$ is the index of the first selected data point, and $(x_i^t, y_i^t)_{m^{(1)}}$ is the selected point to be stored in memory.

IPM also relies on the well learned features of each class for best performance in sample selection and thus must be performed at the conclusion of a task. This also makes IPM most suitable for the offline continual learning scenario.

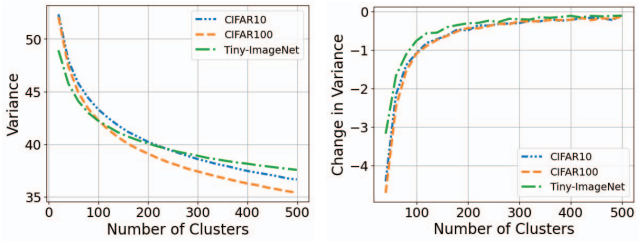
Similar to herding, we again store $|\mathcal{M}|/s$ samples per class for a balanced and saturated buffer and preserve the most informative samples, as determined by IPM, by deleting the most recently added samples to each class when new data is encountered and must be stored to the fixed buffer.

3.5. Dynamic Memory Buffers

Traditionally, when replay methods are used, it is common for the buffer size to be fixed [32, 5, 4]. This fixed buffer size does not take into account the underlying complexity of the data and more specifically, each class within the data. To account for such dataset specific complexities, we allow the buffer to be dynamic, where we add K samples of class c to the buffer using two algorithm agnostic [1] methods we refer to as intracluster variance and Kaiser criterion described below. This idea is motivated by providing a guideline for determining the number of samples needed to represent the data manifold and the different classes within it as opposed to arbitrarily choosing a fixed buffer size.

Intracluster Variance. We assume that the number of images necessary for replay depends on the complexity of the underlying manifold of the data. It is well known that relations between the data points on the high dimensional manifold are preserved when the data is embedded into a lower dimension via norm preserving transformations [6, 12]. For example, the dominant eigenvectors (or principal components) of the space in which the images lie is an

[1] Algorithm agnostic refers to each population strategy as each strategy will choose different subsets of samples bounded by each studied dynamic buffer criterion.



(a) (b)

Figure 1: (a) Average intracluster variance for varying number of clusters. (b) Average change in intracluster variance for varying number of clusters.

example of one such transformation. In fact, the number of eigenvectors needed to represent the data is an indication of the complexity of the manifold, and therefore provides some indication of the number of images required for replay. Another method for estimating the complexity of the manifold is by forming clusters in the data and by observing the change in the average intracluster variance for each cluster.

Assume that the data is grouped into K clusters, each with N_j images. Specifically, let $x_{i,j}$ for $1 \leq i \leq N_j$, $1 \leq j \leq K$ represent the i^{th} training image of the j^{th} cluster. The variance of any given cluster is given by $\sigma_j^2 = \frac{1}{N_j}\sum_{i=1}^{N_j}(x_{i,j} - \mu_j)$ where $\mu_j = \frac{1}{N_j}\sum_{i=1}^{N_j} x_{i,j}$ is the mean of the respective cluster. The average variance of all clusters is then simply $\sigma_K^2 = \frac{1}{K}\sum_{j=1}^{K}\sigma_j^2$. The premise is that each cluster represents a local region on the manifold where the data is concentrated. As K increases, each cluster becomes more and more compact and their variance σ_j^2 for $1 \leq j \leq K$ decreases. This causes the average variance σ_K^2 to also decrease as shown in Figure 1a using the CIFAR10, CIFAR100 and Tiny-ImageNet datasets [20, 21] and the k-means clustering algorithm. In fact, the change in variance $\Delta_K = \sigma_{K+1}^2 - \sigma_K^2$ also tapers asymptotically as K is allowed to increase. Figure 1b shows this behavior and the rate at which the average variance decreases.

We observe that increasing the number of clusters beyond the knee of the curve, 100 for both CIFAR10 and CIFAR100 and 160 for Tiny-ImageNet, provides diminishing gains in terms of decreasing the variance of each cluster. We assume that each cluster is as compact as possible, and the cluster mean is a good representation of the data that lies within the cluster. Therefore, we choose the number of clusters where the knee of the curve (found using the Kneedle algorithm [34]) in 1a and 1b occur as an indication of the number of representative samples that are required to adequately represent the data manifold.

With this estimate of the number of samples needed for proper representation of the underlying data manifold, we populate the buffer with K samples per class. This allows

Dataset	Mean	Min.	Max
CIFAR10	248 ± 27	215	290
CIFAR100	50 ± 8	28	69
TinyImageNet	59 ± 9	41	94

Table 1: Kaiser Criterion stats for number of samples required for each class.

for the buffer to maintain its class balanced properly and avoid biased sampling from the buffer when training.

When using instracluster variance, one must have access to the entire dataset before training begins. This can be seen as unfair in the context of continual learning since in real world scenarios it would be impossible to view the entire dataset beforehand. We address this concern by using the Kaiser criterion below.

Kaiser Criterion. Instead of finding a global number of samples to be kept for each class as done in intracluster variance, we detail an additional method based on the number of most useful eigenvectors for each class.

Assume that $\mathbf{x}_{i,c}$ is a d dimensional column vector that represents the i^{th} training image of class c for $1 \leq i \leq N_c$. For each class, we define the data matrix $\boldsymbol{A}_c = [\mathbf{x}_{1,c}, ..., \mathbf{x}_{1,N_c}]$, and compute the eigenvalues and eigenvectors of $\boldsymbol{A}_c \boldsymbol{A}_c^T$ denoted as $\lambda_{k,c}$ and $\boldsymbol{\phi}_{k,c}$, respectively, for $k = 1, ..., d$. Loosely speaking, the rank of this matrix is a proxy for the number of independent images required to represent the dataset and can thus be viewed as a reduced version of PCA. This in turn is related to the non-singular eigenvalues. Therefore, to determine the minimum number of images required to represent the data, we count the number of non-trivial eigenvalues (ignoring the ones that are zero or close to it). The Kaiser criterion [18] is a common method for selecting the useful eigenvectors which states that only those with eigenvalues greater than 1.0 should be retained for representing the data.

We note that this method of using raw images to form the data matrix is better suited than using network features, as that would always require the network to accurately classify the entire data, which manot not occur if a particular class is difficult for the network to classify.

We report the statistics of the Kaiser criterion for each dataset in Table 1. The benefits of using a dynamic buffer with the Kaiser criterion allow for classes with higher complexity to have more representation in the buffer, which ultimately leads to higher probability of sampling these more complex classes from the buffer and further mitigate forgetting.

Additionally, the Kaiser criterion can be used in an offline manner before the training of each task as the determination for the number of samples to be stored for each class depends only on a specific class's data matrix. Thus, when new task data is available, we can compute the Kaiser criterion for each individual class by partitioning the task data matrix into class data matrices.

4. Experiments

We investigate the performance of each population strategy described above by comparing each scheme under a commonly used suite of replay-based training methods. We test using both a fixed memory buffer with commonly used buffer sizes and the newly proposed dynamic buffer scheme described in Section 3.5. Our primary focus is on the offline class-IL setting. We report offline task-IL results in the supplementary material.

Datasets. We benchmark each population strategy on three commonly used continual learning datasets: split-CIFAR10, split-CIFAR100 and split-Tiny-ImageNet [20, 45, 21]. In split-CIFAR10, the CIFAR10 dataset is split into 5 disjoint tasks where each task contains 2 classes. Split-CIFAR100 splits CIFAR100 into 10 disjoint tasks of 10 classes each. Split-Tiny-ImageNet splits Tiny-ImageNet into 10 disjoint tasks of 20 classes each. We maintain the same order in which classes are split across all tested population strategies and methods.

Compared Methods. Each population strategy studied herein is tested against four commonly compared methods in continual learning literature, namely ER [32], DER [4], GDumb [28], and ER-ACE [5]. Each of the aforementioned methods was state of the art for its time with ER-ACE being the most recently proposed state of the art method for bench-marking of current replay-based continual learning research.

All compared methods use some form of experience replay. For each $t > 1$, all methods train on the union of task data and data stored in memory as $\mathcal{D}_t \cup \mathcal{M}$, except in the case of GDumb where we only populate $\mathcal{M}$ in a balanced manner by performing one iteration through each $\mathcal{D}_t$ and proceed to train solely on $\mathcal{M}$ at the conclusion of the final task.

Configuration and Hyperparameters. We test each of the buffer population strategies studied herein with the above described methods using the open-source codebase, *Mammoth*, first introduced in [4]. We use the best configuration for each compared training method when testing performance of each population strategy with a ResNet18 [14] backbone for fair comparisons. Exact hyperparmeter configuration can be found in the supplementary material.

Metrics. We judge performance with the commonly used metrics of final average accuracy (*FAA*) and final forgetting (*FF*) given by

$$FAA = \frac{1}{\mathcal{T}} \sum_{j=1}^{\mathcal{T}} a_j^{\mathcal{T}} \tag{6}$$

Fixed Buffer Size	Method	Population Strategy	Split-CIFAR10 Class-IL		Split-CIFAR100 Class-IL		Split-TinyImageNet Class-IL	
			FAA	*FF*	*FAA*	*FF*	*FAA*	*FF*
200	ER	Reservoir	48.39 ± 2.01	60.39 ± 2.26	15.35 ± 0.86	81.20 ± 0.62	8.40 ± 0.16	76.88 ± 0.18
		Herding	**52.32 ± 0.77**	**55.17 ± 0.77**	**15.94 ± 0.46**	**79.60 ± 0.11**	**8.81 ± 0.05**	**76.61 ± 0.62**
		GSS	41.45 ± 4.65	68.92 ± 5.57	11.87 ± 0.05	84.34 ± 0.41	-	-
		IPM	48.68 ± 1.11	59.44 ± 1.32	15.0 ± 0.26	80.67 ± 0.31	8.55 ± 0.09	77.13 ± 0.44
	DER	Reservoir	**61.17 ± 1.08**	41.27 ± 0.53	24.38 ± 1.46	70.07 ± 1.76	11.15 ± 0.46	74.22 ± 0.65
		Herding	30.01 ± 1.87	83.12 ± 2.01	9.99 ± 0.23	87.73 ± 0.50	5.7 ± 1.06	71.92 ± 1.87
		GSS	38.04 ± 6.09	71.95 ± 7.60	12.29 ± 1.70	78.05 ± 0.53	-	-
		IPM	60.48 ± 0.34	**30.12 ± 1.03**	**33.47 ± 1.86**	**42.32 ± 1.71**	**19.36 ± 1.06**	**45.46 ± 0.63**
	GDumb	Reservoir	29.26 ± 1.18	N/A	4.63 ± 0.49	N/A	2.13 ± 0.28	N/A
		Herding	**32.16 ± 1.51**	N/A	**7.02 ± 0.51**	N/A	**3.45 ± 0.30**	N/A
		GSS	28.35 ± 1.19	N/A	4.81 ± 0.29	N/A	-	N/A
		IPM	31.60 ± 1.83	N/A	6.26 ± 0.14	N/A	2.67 ± 0.36	N/A
	ER-ACE	Reservoir	**63.32 ± 2.40**	18.62 ± 2.24	28.78 ± 0.66	44.40 ± 1.13	12.82 ± 0.11	48.91 ± 1.65
		Herding	61.66 ± 0.35	33.04 ± 3.37	**29.64 ± 0.29**	62.08 ± 0.27	**15.75 ± 0.09**	64.86 ± 1.45
		GSS	26.25 ± 9.53	**1.02 ± 1.01**	7.525 ± 0.13	**8.93 ± 0.75**	-	-
		IPM	49.41 ± 0.56	16.56 ± 0.09	28.36 ± 0.28	27.17 ± 0.23	15.02 ± 0.53	**29.16 ± 0.35**
500	ER	Reservoir	61.07 ± 0.64	44.27 ± 1.10	21.37 ± 1.27	73.55 ± 0.77	10.19 ± 0.20	75.34 ± 0.16
		Herding	**64.09 ± 0.73**	**39.26 ± 1.39**	**24.25 ± 0.70**	**70.46 ± 0.57**	**10.38 ± 0.16**	75.50 ± 0.35
		GSS	61.07 ± 0.64	44.27 ± 1.10	21.37 ± 1.27	73.55 ± 0.77	-	-
		IPM	61.08 ± 0.46	44.33 ± 0.68	22.06 ± 0.48	72.80 ± 0.33	10.20 ± 0.17	**74.98 ± 0.18**
	DER	Reservoir	**70.07 ± 0.95**	29.5 ± 1.80	34.53 ± 1.68	56.30 ± 1.52	17.15 ± 1.40	66.91 ± 1.81
		Herding	48.20 ± 2.94	54.64 ± 7.93	13.11 ± 0.51	84.04 ± 0.69	5.32 ± 0.78	67.13 ± 2.02
		GSS	45.94 ± 6.22	59.09 ± 9.16	16.64 ± 1.69	72.323 ± 5.59	-	-
		IPM	65.5 ± 2.68	**23.55 ± 5.19**	**40.51 ± 0.43**	**28.97 ± 1.06**	**20.49 ± 0.86**	**31.54 ± 2.61**
	GDumb	Reservoir	**43.35 ± 0.55**	N/A	9.85 ± 0.45	N/A	3.6 ± 0.004	N/A
		Herding	42.85 ± 0.83	N/A	**11.45 ± 0.42**	N/A	**4.83 ± 0.19**	N/A
		GSS	37.39 ± 1.21	N/A	6.2 ± 0.29	N/A	-	N/A
		IPM	42.03 ± 3.05	N/A	9.02 ± 0.67	N/A	3.27 ± 0.40	N/A
	ER-ACE	Reservoir	**72.15 ± 0.38**	13.18 ± 1.26	**37.60 ± 0.15**	38.17 ± 1.16	**20.99 ± 0.52**	46.60 ± 0.55
		Herding	69.75 ± 1.90	23.00 ± 4.32	37.54 ± 0.44	53.08 ± 0.47	20.3 ± 0.36	60.48 ± 0.06
		GSS	19.54 ± 0.09	**0.22 ± 0.11**	8.34 ± 0.09	**7.73 ± 0.62**	-	-
		IPM	54.43 ± 0.99	14.57 ± 1.17	35.01 ± 0.14	25.50 ± 0.91	18.5 ± 0.42	**34.41 ± 8.08**
5120	ER	Reservoir	83.18 ± 1.82	14.33 ± 1.65	50.71 ± 0.27	38.92 ± 0.44	27.36 ± 0.03	54.46 ± 0.69
		Herding	**85.56 ± 0.34**	12.22 ± 0.29	**52.93 ± 1.55**	**35.21 ± 0.93**	**28.43 ± 0.51**	**52.12 ± 0.51**
		GSS	60.35 ± 7.06	43.74 ± 8.69	17.52 ± 0.22	78.74 ± 2.28	-	-
		IPM	85.19 ± 0.63	**10.97 ± 0.91**	50.75 ± 1.16	36.96 ± 0.60	27.39 ± 0.23	53.43 ± 0.12
	DER	Reservoir	**83.35 ± 0.72**	11.27 ± 0.96	57.22 ± 0.24	22.86 ± 1.51	**37.09 ± 0.50**	31.91 ± 1.05
		Herding	76.21 ± 1.08	25.26 ± 1.63	52.53 ± 0.24	38.70 ± 0.38	5.09 ± 2.59	38.44 ± 5.37
		GSS	45.86 ± 16.34	54.54 ± 21.02	56.32 ± 1.10	45.05 ± 7.04	-	-
		IPM	67.75 ± 0.52	**6.25 ± 1.21**	**57.30 ± 0.51**	**8.85 ± 0.52**	34.33 ± 0.93	**7.27 ± 0.98**
	GDumb	Reservoir	**79.89 ± 1.29**	N/A	**42.52 ± 0.22**	N/A	**21.18 ± 0.06**	N/A
		Herding	77.16 ± 0.74	N/A	36.80 ± 0.57	N/A	17.38 ± 0.40	N/A
		GSS	70.27 ± 2.29	N/A	19.64 ± 1.40	N/A	-	N/A
		IPM	79.58 ± 0.93	N/A	42.31 ± 0.18	N/A	20.99 ± 0.72	N/A
	ER-ACE	Reservoir	83.67 ± 0.26	4.93 ± 0.30	57.01 ± 0.27	21.02 ± 0.22	**38.68 ± 0.37**	29.41 ± 0.74
		Herding	**85.02 ± 0.86**	11.83 ± 2.08	**58.52 ± 0.23**	26.77 ± 0.19	34.66 ± 0.99	41.91 ± 1.08
		GSS	19.73 ± 0.01	**0.05 ± 0.02**	9.20 ± 0.04	**4.31 ± 0.71**	-	-
		IPM	66.69 ± 0.46	6.29 ± 0.38	53.91 ± 0.43	14.69 ± 0.46	36.28 ± 0.19	**18.99 ± 0.27**

Table 2: Population strategy results tested with various replay based methods with traditionally used fixed size buffer, averaged across three runs. We do not report forgetting in GDumb experiments due to the nature of GDumb only training on the fully populated, balanced buffer. Results for TinyImageNet are not reported for GSS due to intractable train times.

$$FF = \frac{1}{\mathcal{T}-1}\sum_{j=1}^{\mathcal{T}-1} f_j^{\mathcal{T}} \text{ s.t. } f_j^{\mathcal{T}} = \max_{l\in\{1,\dots,\mathcal{T}-1\}} a_l^l - a_j^{\mathcal{T}} \quad (7)$$

where $a_j^{\mathcal{T}}$ and $f_j^{\mathcal{T}}$ are interpreted as the accuracy and forgetting of task j and the end of training on $\mathcal{T}$ tasks respectively [8]. When judging performance, we seek maximal *FAA* and minimal *FF*.

4.1. Results

Fixed Buffer. Results for commonly tested fixed buffer sizes are in Table 2. We first observe that in nearly all cases, reservoir sampling leads to greater forgetting when compared to the other population strategies, specifically compared to herding and IPM. We also observe in several trials, that reservoir sampling also underperforms in *FAA* compared to herding and IPM. This behavior can be attributed to both herding and IPM selecting the best samples from the learned feature space at the conclusion of each task whereas reservoir sampling simply selects and replaces samples at random.

Another reason for greater forgetting in reservoir sam-

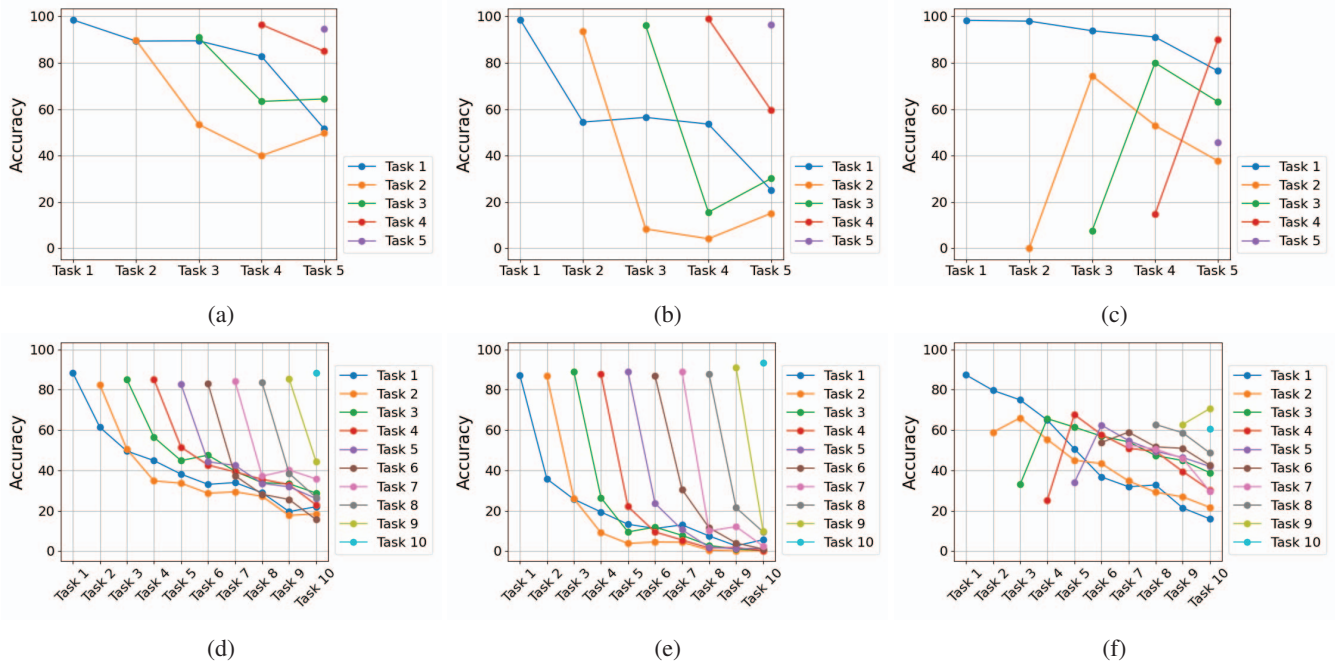


Figure 2: A comparison of the reservoir, herding, and IPM population strategies paired with DER with a fixed buffer size of 500. Top row corresponds to Split-CIFAR10 performance and bottom row is Split-CIFAR100. The columns correspond as follows: left uses reservoir sampling, center uses herding, and right uses IPM. We do not report GSS results due to all around inferior performance.

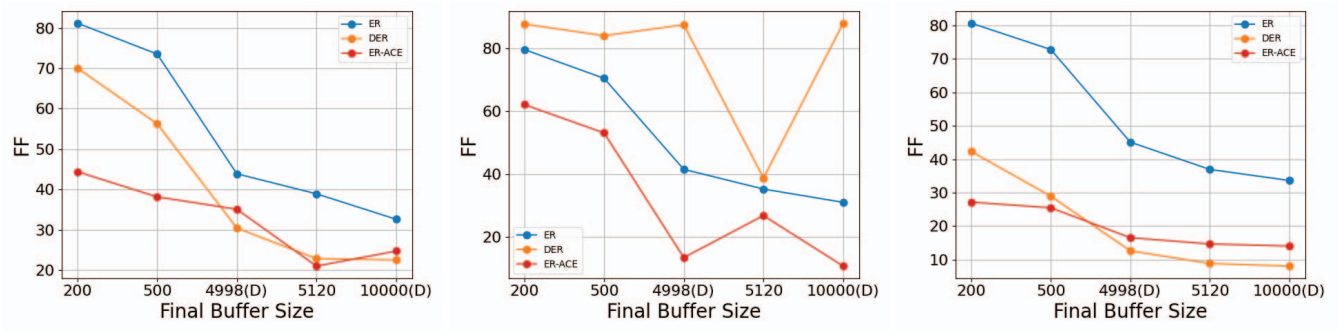
Fixed Buffer Size	Percentage of Samples Belonging to Each Task ($t_1/t_2/t_3/t_4/t_5$)
200	23.5% / 19.0% / 16.0% / 22.0% / 19.5%
500	23.5% / 15.0% / 20.5% / 22.5% / 18.5%
5120	20.0% / 18.5% / 19.53% / 20.43% / 21.54%

Table 3: Percentage of samples in buffer belonging to each task at the end of training populated via reservoir sampling.

pling is the unbalanced fixed buffers incurred by the random sampling and replacement of data in the buffer. We show the percentage of task specific data of the final fixed buffer when populated via reservoir sampling in Table 3. We can clearly see that, for smaller buffer sizes, there is a bias to the storage of earlier task data compared to more recently encountered tasks. In comparison, both herding and IPM maintain a balanced fixed buffer at all times leading to equal probability of sampling any task data for batch training. Naturally, as the buffer size increases, the unbalanced nature of the buffer populated with reservoir sampling becomes less severe and we see the reservoir population strategy become more competitive with herding and IPM in both *FAA* and *FF*.

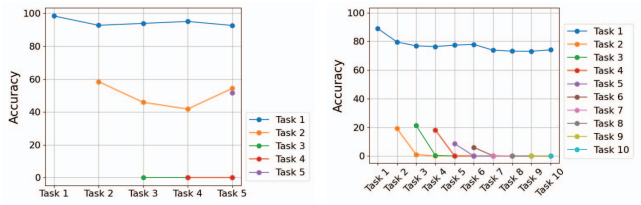
We pay particular interest to the scenarios where IPM yields superior *FF* yet inferior *FAA*. To analyze why this happens in certain cases, we plot each population strategy used in conjunction with DER for both split-CIFAR10 and split-CIFAR100 in Figure 2 (we omit GSS figures due to all around inferior performance). We observe the interesting behavior where for each $t > 1$, IPM initially has poor current task performance but then proceeds to make astonishing recoveries for each subsequent task. This indicates that IPM tends to prioritize performance on the buffer instead of the current task at hand which in turn leads to lesser forgetting. An interesting observation to make is that this behavior holds only for DER and ER-ACE (see supplementary material for additional figures). Both DER and ER-ACE are training schemes that directly optimize on logits indicating that IPM is best suited for these types of training schemes.

We take note of the relatively low forgetting of the GSS population strategy when tested with ER-ACE. To investigate why GSS achieves such low forgetting, we plot the accuracy of each previously learned task as new tasks are learned in Figure 4. From this, we can infer that GSS's low forgetting capabilities when coupled with ER-ACE is caused by the inferior performance on subsequent tasks after $t = 1$ and thus, has nearly nothing to forget. We note the strong performance of $t = 1$ throughout the model's life, however. This is due to GSS first populating the buffer with $t = 1$ data and thereafter, hardly ever finding a sam-



(a) (b) (c)

Figure 3: Final forgetting performance with various final buffer sizes tested with Split-CIFAR100. Final buffer sizes with a (D) indicate dynamic final size. In order from left to right are results from reservoir, herding, and IPM respectively.



(a) (b)

Figure 4: ER-ACE using the GSS population strategy for a fixed buffer with buffer size 200. (a) Split-CIFAR10 results. (b) Split-CIFAR100 results.

ple with an appropriate score to replace other samples in the buffer as described in Section 3.3. Because of this nature, we should not accept that GSS is the best forgetting performer with coupled with ER-ACE. We still observe that reservoir sampling mostly does not compare in forgetting to other strategies when ignoring GSS results.

We observe the all around competitiveness of each population strategy when tested with GDumb (note, we do not report *FF* for GDumb as there is no forgetting to take place since GDumb trains solely on the buffer). Because GDumb only uses each observed task to populate the memory buffer, it makes sense that herding and IPM perform roughly the same with reservoir sampling since herding and IPM depend on the well learned features for population. Similarly, GSS depends on the gradients of each sample, but because GDumb takes no gradient steps until the conclusion of the final observed task, GSS has no proper way to score samples for replacement or not.

Lastly, we make the note that while no single method consistently outperforms any other, we demonstrate that in many situations, reservoir sampling yields inferior final forgetting performance. This suggests that when using replay-based methods in continual learning solutions, reservoir sampling should not be blindly used as the buffer population strategy of choice, and one should instead pay careful attention to selection of buffer population algorithm for the best performance.

Dynamic Buffer. We next perform experiments using dynamic buffers using the two criteria as described in Section 3.5 and provide tabulated results for class-IL and task-IL scenarios in the supplementary material. We omit GSS results using a dynamic buffer due to poor performance with paired with any of the fixed buffer schemes.

Overall, we observe much of the same trends as seen with fixed buffers. However, we notice that dynamic buffers seem to benefit most when paired with reservoir sampling and IPM in Figure 3 (*FAA* and *FF* for all datasets are reported in supplementary material). Because dynamic buffers find the optimal number of samples for storage, we expect the change in *FF* to be lower when we approach that number, which we observe in Figure 3. In general we observe the Kaiser criterion performing better than intracluster variance. This can be attributed to the ability to adapt to classs complexity for the Kaiser criterion, particularly as the number of classes increases in a dataset.

We note the curious performance of DER when coupled with dynamic replay and give a brief conjecture for why this may be in the supplemental material.

5. Conclusions

In this work, we compare the commonly used approach of reservoir sampling for memory buffer population in replay methods to other greedy sampling methods to answer the question of *which samples should be stored* in memory. We show that reservoir sampling tends to lead to higher forgetting when compared to methods that use strategic population strategies to select the best data points for storage. We then address the question of *how many samples should be stored* by the formulation of a dynamic buffer populated according to two criteria based on a dataset's eigenvectors and eigenvalues. We show that dynamic buffers lead to more competitive performance for all population strategies when

compared to the arbitrary fixed buffer sizes commonly used in replay methods.

References

[1] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019. 1, 2

[2] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019. 1, 3

[3] Lorenzo Bonicelli, Matteo Boschini, Angelo Porrello, Concetto Spampinato, and Simone Calderara. On the effectiveness of lipschitz-driven rehearsal in continual learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 1, 3

[4] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020. 1, 2, 4, 5

[5] 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 *International Conference on Learning Representations*, 2022. 1, 2, 4, 5

[6] Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005. 4

[7] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 9516–9525, 2021. 2

[8] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European conference on computer vision (ECCV)*, pages 532–547, 2018. 6

[9] Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6993–7001, 2021. 2

[10] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*, 2019. 2

[11] Ehsan Elhamifar, Guillermo Sapiro, and René Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1600–1607, 2012. 3

[12] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016. 4

[13] Tyler L Hayes, Giri P Krishnan, Maxim Bazhenov, Hava T Siegelmann, Terrence J Sejnowski, and Christopher Kanan. Replay in deep learning: Current approaches and missing biological elements. *Neural computation*, 33(11):2908–2950, 2021. 1

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[15] Mohsen Joneidi, Saeed Vahidian, Ashkan Esmaeili, Weijia Wang, Nazanin Rahnavard, Bill Lin, and Mubarak Shah. Select to better learn: Fast and accurate deep learning using data selection from nonlinear manifolds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 4

[16] Mohsen Joneidi, Alireza Zaeemzadeh, Nazanin Rahnavard, and Mubarak Shah. Iterative projection and matching: Finding structure-preserving representatives and its application to computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5409–5418, 2019. 3, 4

[17] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetful learning for domain expansion in deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 2

[18] Henry F Kaiser. The application of electronic computers to factor analysis. *Educational and psychological measurement*, 20(1):141–151, 1960. 5

[19] 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

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 4, 5

[21] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015. 4, 5

[22] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017. 2

[23] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems*, 30, 2017. 2

[24] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995. 1

[25] 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

[26] Nikhil Mehta, Kevin Liang, Vinay Kumar Verma, and Lawrence Carin. Continual learning using a bayesian nonparametric dictionary of weight factors. In *International Conference on Artificial Intelligence and Statistics*, pages 100–108. PMLR, 2021. 2

[27] Quang Pham, Chenghao Liu, and Steven HOI. Dualnet: Continual learning, fast and slow. In A. Beygelzimer, Y.

Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. 1, 2

[28] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 524–540. Springer, 2020. 5

[29] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 1, 2, 3

[30] Mark B Ring. Child: A first step towards continual learning. In *Learning to learn*, pages 261–292. Springer, 1998. 1

[31] Anthony V. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connect. Sci.*, 7:123–146, 1995. 2

[32] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2, 4, 5

[33] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 2

[34] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, 2011. 4

[35] Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. *Advances in Neural Information Processing Systems*, 33:15579–15590, 2020. 2

[36] Qing Sun, Fan Lyu, Fanhua Shang, Wei Feng, and Liang Wan. Exploring example influence in continual learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 1

[37] Shengyang Sun, Daniele Calandriello, Huiyi Hu, Ang Li, and Michalis Titsias. Information-theoretic online memory selection for continual learning. In *International Conference on Learning Representations*, 2022. 1

[38] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995. 1

[39] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019. 1

[40] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. 3

[41] Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020. 2

[42] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009. 3

[43] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. In *International Conference on Learning Representations*, 2022. 1

[44] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018. 2

[45] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017. 5

[46] A. Çivril. Column subset selection problem is ug-hard. *Journal of Computer and System Sciences*, 80(4):849–859, 2014. 4