Online Continual Learning from Imbalanced Data

Aristotelis Chrysakis [1] Marie-Francine Moens [1]

Abstract

A well-documented weakness of neural networks is the fact that they suffer from *catastrophic forgetting* when trained on data provided by a non-stationary distribution. Recent work in the field of *continual learning* attempts to understand and overcome this issue. Unfortunately, the majority of relevant work embraces the implicit assumption that the distribution of observed data is perfectly balanced, despite the fact that, in the real world, humans and animals learn from observations that are temporally correlated and severely imbalanced. Motivated by this remark, we aim to evaluate memory population methods that are used in *online* continual learning, when dealing with highly imbalanced and temporally correlated *streams* of data. More importantly, we introduce a new memory population approach, which we call *class-balancing reservoir sampling* (CBRS). We demonstrate that CBRS outperforms the state-of-the-art memory population algorithms in a considerably challenging learning setting, over a range of different datasets, and for multiple architectures.

1. Introduction

Over the past decade, deep neural networks have been used to tackle an impressive range of problems. Such problems vary from classifying (Krizhevsky et al., 2012) or generating images (Radford et al., 2015) to translating natural language (Bahdanau et al., 2015) and outperforming humans at playing several board games (Silver et al., 2018). The models utilized to solve the aforementioned tasks are typically trained offline by performing multiple passes over large amounts of previously collected (mostly labeled) data.

The accumulation of knowledge and experience by humans is a vastly different story. Over our lives we perceive a *stream* of temporally correlated, unlabeled observations, and rarely revisit the same observation multiple times (Parisi et al., 2019). Moreover, the learning and application of new knowledge takes place concurrently, rather than in two distinct stages, by interacting with our surrounding environment (Tani, 2016). Finally, humans learn to address a large number of problems (i.e., visual understanding of our surroundings, communication, use of our limbs etc.), instead of only just one (Cangelosi & Schlesinger, 2015).

In general, neural networks perform poorly when trained on a non-stationary distribution of data. Assuming a network is being trained on a sequence of distinct tasks, sometimes called a *continuum*, it learns the currently presented task but fails to remember previously learned ones (Goodfellow et al., 2014). This phenomenon is defined as *catastrophic forgetting* (McCloskey & Cohen, 1989; French, 1999).

The field researching how to overcome catastrophic forgetting is called *continual learning* (CL). Two distinct CL settings have primarily been investigated (De Lange et al., 2019). The first one, termed *task-incremental* CL, assumes that each task of the continuum can be learned offline. Specifically, the learner is given access to all the data from the current task and can perform numerous passes over it. Under the second CL setting, which is being referred to as *online* CL, the learner is presented with a stream of tiny batches of observations and cannot revisit previously seen batches from the current or the previous tasks.

A majority of recent work has centered on the task-incremental setting. In this work, however, we focus on online CL, as it more closely resembles the way humans and animals learn (Cangelosi & Schlesinger, 2015). In particular, we focus on replay-based online CL, under which a small part of the incoming stream of observations are memorized, so that they can be used to retrain the learner in the future. Research in the field of *neuroscience* has shown that replay is also present in living beings and is connected to the process of memory consolidation (Girardeau et al., 2009; Ego-Stengel & Wilson, 2010).

Our main motivation has been to emphasize a gap that exists in CL research. Specifically, most of the benchmarks used to evaluate CL approaches are perfectly balanced, both in terms of the sizes of the different tasks, and in terms of the

[1]Department of Computer Science, KU Leuven, Leuven, Belgium. Correspondence to: Aristotelis Chrysakis <aristotelis.chrysakis@kuleuven.be>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

classes that comprise each task. Consequently, the majority of the proposed approaches assume either explicitly or implicitly that the data used to train the model is perfectly balanced. In contrast, a living being will experience major imbalances in the experiences it acquires over its lifetime.

Here, we consider an online CL setting that contains significant data imbalances and does not provide task identifiers or boundaries. To tackle such settings, we propose *class-balancing reservoir sampling* (CBRS), a novel memory population approach, which does not require any prior knowledge about the incoming stream and does not make any assumptions about its distribution. CBRS is designed to be able to balance the stored data instances with respect to their class labels, having made only one pass over the stream. We compare CBRS to state-of-the-art memory population methods, over four different datasets, and two different neural network architectures. We demonstrate that it consistently outperforms the state of the art (with relative differences of up to more than 40%), while being equally or more efficient computationally. We present a sizeable amount of experimental results that support our claims.

1.1. Document Structure

In Section 2 we present the main techniques proposed in this work — namely, the CBRS algorithm and the use of weighted replay. In Section 3 we describe our experimental work and discuss the corresponding results. In Section 4 we review other work relevant to ours, and finally, in Section 5, we offer our concluding remarks.

1.2. Notation

We use lower-case boldface characters to denote a vector (e.g., $\mathbf{v}$) and lower-case italic characters for scalars (e.g., s), with the only exception being loss values, where the special symbol $\mathcal{L}$ is used . Upper-case boldface characters signify a matrix (e.g., $\mathbf{M}$).

2. Methodology

2.1. Online Continual Learning

Online continual learning can be formally defined as learning from a lengthy stream of data that is produced by a non-stationary distribution (Aljundi et al., 2019a). The stream is generally modeled as a sequence of distinct *tasks*, each of them containing instances from a specific set of classes (Chaudhry et al., 2019a). The data distribution is typically assumed to be stationary throughout each task (Aljundi et al., 2019a). The stream provides the learner with data instances $(\mathbf{x}_t, y_t)$ or small batches of them $(\mathbf{X}_t, \mathbf{y}_t)$, where $\mathbf{x}_t, y_t$ are the input and label of the instance respectively, and t refers to the current time-step. An incoming instance or batch cannot be revisited once the next instance or batch is received (Chaudhry et al., 2019a). A more informative stream might also provide the learner with so-called *task boundaries*, hinting that the current task has been completed and the next one is about to start.

The learner is typically evaluated after the stream has been received in its entirety. During the evaluation process, some learners require *task identifiers* — that is, information that communicates to them the task to which the instances to be predicted belong, before they make their prediction (Lopez-Paz & Ranzato, 2017). Having access to task identifiers simplifies CL significantly (van de Ven & Tolias, 2019), but is inconsistent with human learning.

In this work, we consider a minimal-prerequisite scenario, where task identifiers and boundaries are not provided during training and evaluation. Additionally, we assume the absence of any prior knowledge regarding the incoming stream (i.e., length, task composition etc.). Our motivation for making these choices is to constrain the learning process so that it approximates the human aggregation of experiences and knowledge.

2.2. Class-Balancing Reservoir Sampling

In this section we describe in detail the main contributions of this work. We propose an algorithm called *class-balancing reservoir sampling* (CBRS) that decides which data instances of the input stream to store for future replay. In essence, having read the stream in its entirety, the aim of CBRS is to store an *independent and identically distributed* (iid) sample from each class, and keep the classes as balanced as possible. In other words, the algorithm preserves the distribution of each class, while at the same time altering the distribution of the stream so that class imbalances are mitigated.

Let us first describe our notation. Given a memory of size m, we will say that the memory is *filled* when all its m storage units are occupied. When a certain class contains the most instances among all the different classes present in the memory, we will call it the *largest*. Two or more classes can be the *largest*, if they are equal in size and also the most numerous. Finally, we will call a class *full* if it currently is, or has been in one of the previous time steps, the *largest* class. Once a class becomes *full*, it remains so in the future. Such classes are called *full*, because CBRS does not allow them to grow in size.

As in the case of reservoir sampling (Vitter, 1985), the CBRS sampling scheme can be split into two stages. During the first stage, and as long as the memory is not *filled*, all the incoming stream instances are getting stored in memory. After the memory is *filled*, we move on to the second stage.

During this stage, when a stream instance $(\mathbf{x}_i, y_i)$ is re-

Algorithm 1 Class-Balancing Reservoir Sampling

```
1: input: stream: ${(\mathbf{x}_i, y_i)}_{i=1}^n$
2: for $i = 1$ to $n$ do
3:   if memory is not filled then
4:     store $(\mathbf{x}_i, y_i)$
5:   else
6:     if $c \equiv y_i$ is not a full class then
7:       find all instances of the largest class
8:       select from them an instance at random
9:       overwrite the selected instance with $(\mathbf{x}_i, y_i)$
10:    else
11:      $m_c \leftarrow$ number of currently stored instances of class $c \equiv y_i$
12:      $n_c \leftarrow$ number of stream instances of class $c \equiv y_i$ encountered thus far
13:      sample $u \sim \text{Uniform}(0, 1)$
14:      if $u \leq m_c/n_c$ then
15:        pick a stored instance of class $c \equiv y_i$ at random
16:        replace it with $(\mathbf{x}_i, y_i)$
17:      else
18:        ignore the instance $(\mathbf{x}_i, y_i)$
19:      end if
20:    end if
21:  end if
22: end for
```

ceived, the algorithm checks at first whether y_i belongs to a *full* class. If it does not, the received instance is stored in the place of another instance that belongs to the *largest* class. This is the component of CBRS that mitigates class imbalances in memory. In the opposite case, the received instance replaces a randomly selected stored instance of the same class c with probability ${}^{m_c}/_{n_c}$, where m_c is the number of instances of the class c currently stored in memory, and n_c the number of stream instances of class c that we have encountered so far. We sketch the pseudocode for the CBRS algorithm in Algorithm 1.

In Figure 1 we compare the memory distributions of CBRS and two state-of-the-art memory population algorithms (Vitter, 1985; Aljundi et al., 2019b)[1]. The input sequence is an imbalanced stream of the first five (for illustrative purposes) classes of the MNIST dataset (LeCun et al., 2010). Evidently, the resulting distributions of stored instances under reservoir sampling and GSS are significantly influenced by the stream distribution. In contrast, CBRS stores all instances of the two smallest classes (classes 0 and 2) and balances the remaining three.

Storing a balanced subset of the input stream for future replay, represents a prior belief that all the observed classes are equally difficult and equally important to learn. This principle is applied because of the absence of any more specific knowledge about the contents of the stream. Nonetheless, if one possesses such knowledge, it is trivial to extend CBRS to take it into account. We describe how in the supplementary material.

As we show in Section 3, CBRS exhibits superior performance compared to the state of the art. At this point, however, we would like to highlight two important properties that CBRS possesses. Assuming a stream that contains instances of n_c distinct classes, and a memory of size m, the following two statements stand.[2]

First, if the stream contains a class with less than ${}^{m}/_{n_c}$ instances, then all of these instances will be stored in memory after the stream has been read in whole. This is a remarkable attribute of CBRS, which guarantees that not even a single instance of severely underrepresented classes will be discarded. We can see this property illustrated in Figure 1. The stream contains instances from $n_c = 5$ classes, and the memory size is size $m = 1000$. Thus, the two classes of the stream that are composed by less than 200 instances (i.e., classes 0 and 2) are stored in their entirety by CBRS.

Second, the subset of the instances from each class that are stored in memory is iid with respect to the instances of the same class contained in the stream. This is another important characteristic of CBRS, that is also present when performing reservoir sampling. Since we cannot assume that the instances of each class are presented in an iid manner over time, it is desirable that we capture a representative sample from each class in memory.

2.3. Weighted Random Replay

In some cases it is impossible to fully utilize the whole capacity of the memory and keep it balanced at the same time. This is well exemplified by Figure 1. Assuming a memory of size $m = 1000$, and keeping in mind that the smallest class of the stream contains 51 instances, one way to keep the memory balanced would be to store 51 instances from each class. In this case, however, only a quarter of the memory would be utilized, which is very wasteful. Thus, we could fill the entire memory using CBRS (as in Figure 1), and try to mitigate the influence of the resulting imbalance later.

A well-known and effective way to deal with such imbalances is oversampling the minority classes (Branco et al., 2016). In our case, we propose the use of a custom replay sampling scheme, where the probability of replaying a certain stored instance is inversely proportional to the number of stored instances of the same class. In other words, instances of a smaller class will have a higher probability of

[1]For more details on these two algorithms, see Subsection 3.1.

[2]We prove both statements in the supplementary material.

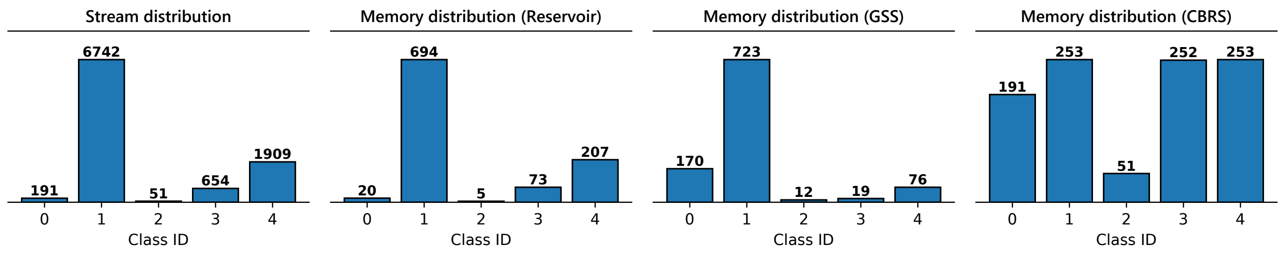

Figure 1. A simple illustration of three memory population methods when learning from an imbalanced stream. All methods employ a memory of size $m = 1000$. We describe the figures from left to right. (i) An imbalanced stream containing instances from the first five classes from MNIST. The resulting memory composition when using Reservoir Sampling (ii) and GSS-Greedy (iii) respectively. Both methods are considerably affected by the distribution of the incoming stream. On the other hand, CBRS (iv) does not discard any instances from the significantly underrepresented classes 0 and 2 and balances the remaining three.

getting replayed, in comparison to instances of a larger one. We will call this scheme *weighted replay*, as opposed to *uniform replay*, under which all stored instances are equally likely to be replayed.

2.4. Putting it All Together

At this point, we describe a general replay-based online CL training process (see Algorithm 2) that is a generalization of previously proposed ones (Chaudhry et al., 2019b; Aljundi et al., 2019a). During the learning process, the model is trained both on the stream, and via replay of the stored instances. The process we describe can be used with any stream, classifier, loss function, memory population algorithm, and replay sampling scheme.

In order to be able to adapt to changes in the stream and at the same time prevent catastrophic forgetting, the model updates are guided by a two-component loss. The first component $\mathcal{L}_s$ is computed with respect to the currently observed stream batch $(\mathbf{X}_t, \mathbf{y}_t)$ of size b, and the second component $\mathcal{L}_r$ with respect to a batch $(\mathbf{X}_r, \mathbf{y}_r)$, also of size b, that was sampled from memory to be replayed. In our implementation, we use as a loss function the cross-entropy between the model predictions and the true labels, but any other differentiable loss could be used. The joint loss is computed as the convex combination of the two loss components

$$\mathcal{L} = a \times \mathcal{L}_s + (1 - a) \times \mathcal{L}_r, \tag{1}$$

where $a \in [0, 1]$ controls the relative importance between them. In practice, a represents a trade-off between quickly adapting to temporal changes in the stream and safeguarding currently possessed knowledge. Previous work either treats the loss components as equally important (Chaudhry et al., 2019b; Aljundi et al., 2019b), or decreases a with the number of completed tasks (Shin et al., 2017; Li & Hoiem, 2017). In our setting, however, we are not provided with task boundaries. We set $a = {}^{1}/{}_{n_c}$, where n_c is the number of distinct classes encountered so far, thus sidestepping the issue of not having access to task information. As we show in the supplementary material, considering the loss components as equally important results in consistently inferior performance.

Algorithm 2 Replay-Based Online Continual Learning

```
1: input: stream: ${(\mathbf{x}_i, y_i)}_{i=1}^n$
2:        model: $f(\cdot)$
3:        loss function: $\ell(\cdot,\cdot)$
4:        batch size: $b$
5:        steps per batch: $n_b$
6: repeat
7:   receive batch $(\mathbf{X}_t, \mathbf{y}_t)$ of size $b$ from stream
8:   $n_c \leftarrow$ number of classes encountered so far
9:   $a \leftarrow 1/n_c$
10:  for $n_b$ steps do
11:    predict outputs: $\hat{\mathbf{y}}_t = f(\mathbf{X}_t)$
12:    stream loss: $\mathcal{L}_s = \ell(\hat{\mathbf{y}}_t, \mathbf{y}_t)$
13:    sample batch $(\mathbf{X}_r, \mathbf{y}_r)$ of size $b$ from memory
14:    predict outputs: $\hat{\mathbf{y}}_r = f(\mathbf{X}_r)$
15:    replay loss: $\mathcal{L}_r = \ell(\hat{\mathbf{y}}_r, \mathbf{y}_r)$
16:    joint loss: $\mathcal{L} = a \times \mathcal{L}_s + (1 - a) \times \mathcal{L}_r$
17:    update model $f$ according to $\mathcal{L}$
18:  end for
19:  for $j = 1$ to $b$ do
20:    $(\mathbf{x}_{t,j}, y_{t,j}) \equiv j$-th instance of batch $(\mathbf{X}_t, \mathbf{y}_t)$
21:    decide whether to store $(\mathbf{x}_{t,j}, y_{t,j})$ according to
       the selected memory population algorithm
22:  end for
23: until the stream has been read in full
```

Note that, although we cannot revisit previously seen batches, we can perform multiple parameter updates on the currently observed batch, as is common practice in online CL (Chaudhry et al., 2019b). In practice, we find that performing just one update per time-step might result in underfitting the data. On the other hand, performing multiple (n_b) updates has an additional computational cost, but is nevertheless beneficial in the sense that it allows for faster adaptation to the non-stationary nature of the stream. Additionally, it permits us to perform multiple replay steps (with a different batch of stored instances each time) for each incoming stream batch, which in turn mitigates forgetting more effectively.

Finally, the learner goes over the data instances contained in the currently observed batch, one at a time, and decides which ones to store, according to the selected memory population algorithm.

3. Experimental Work

In this section, we aim to compare CBRS to the state of the art when it comes to memory population strategies under online CL scenarios. We exclusively consider replay-based training as it is the only one that is suitable to our setting.

3.1. Memory Population Approaches

At the time of writing, reservoir sampling (Vitter, 1985) and gradient-space sampling (Aljundi et al., 2019b) are considered the state-of-the-art approaches at populating the memory during online CL. We refer to these methods as RESERVOIR and GSS respectively.[3]

The memory population strategy that RESERVOIR follows is split in two phases. During the first phase, which lasts until the memory gets filled, all encountered data instances are stored in empty memory spots. In the second phase, which starts once the memory gets filled and continues from then on, the currently observed data instance is stored with probability m/n, where m is the size of the memory and n is the number of data instances encountered so far. The data instance is stored in a memory spot that is uniformly selected, thus all currently stored instances are equally likely to be overwritten. It can be proven that RESERVOIR is equivalent to extracting an iid subset of size m from the stream.

GSS attempts to greedily maximize the variance of the gradient directions of the samples contained in memory (Aljundi et al., 2019b). To that end, it computes a score of similarity between the incoming instance and some randomly sampled stored instances. If the similarity score is small, the instance is more likely to be stored, with instances that have a high score being more likely to get overwritten in the process.

In addition to the aforementioned methods, we consider two weaker baselines — one that trains a model exclusively on the stream without replaying stored memories, and another that stores the current stream instance with 50% probability, in the place of a randomly picked stored instance. We call these methods NAIVE and RANDOM respectively.

[3]Of the two GSS alternatives proposed in Aljundi et al. (2019b), we select the best-performing one, namely GSS-Greedy.

3.2. Simulating Imbalances

In this subsection, we describe our approach to creating imbalanced streams. For each class, we define its *retention factor* as the percentage of its instances in the original dataset that will be present in the stream. We define a vector $\mathbf{r}$ containing k retention factors as follows:

$$\mathbf{r} = (r_1, r_2, \cdots, r_k)\,. \tag{2}$$

We distribute the retention factors to each class at random and without replacement, starting over if the number of classes in the dataset is larger than k. In practice, we use

$$\mathbf{r} = \left(10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}\right) \tag{3}$$

for all of our experiments. Given this choice, the minimum imbalance that can exist between two classes in a stream is approximately three to one, while the maximum is 100 to one. The selection of these retention factors represents a trade-off between simulating extreme imbalances and ensuring adequate class representation. Specifically, we take into account the fact that each class in CIFAR-100 (see Subsection 3.3) contains only 500 instances. Consequently, this selection of $\mathbf{r}$ ensures that at least five instances from each class will be present in the resulting stream, while allowing for imbalances of up to two orders of magnitude.

The retention factors for each class are selected at random in every run, and thus each run is performed using a stream with different imbalances. We want to stress, however, that in each experiment, all memory population methods are compared on exactly the same group of imbalanced streams. We do not discard other sources of stochasticity such as the random initialization of a model or the replay sampling.

3.3. Benchmarks

Datasets Following Aljundi et al. (2019b), we select MNIST (LeCun et al., 2010) and CIFAR-10 (Krizhevsky, 2012) for our experiments. In addition, we use Fashion-MNIST (Xiao et al., 2017) and CIFAR-100 (Krizhevsky, 2012). All four of the datasets used in this work are freely available online. We evaluate each method on the standard test set of each selected dataset.

We opt for using one class per task, with the classes being presented in increasing order. This way, the continuum is more difficult to learn since it is split in more distinct tasks. When allowing for two or more classes to be present and iid in each task, as is usually the case in the relevant research (Chaudhry et al., 2019b; Aljundi et al., 2019b), the stream remains stationary for longer periods of time steps, and is thus easier to learn.

All datasets are used as *split* benchmarks — that is to say, the learner is first presented with the instances of the first class, then with the ones of the second, and so on. We found the influence of the class ordering on the results to be negligible.[4] We intentionally refrain from using the *permuted* MNIST benchmark (Goodfellow et al., 2014), due to the criticism it has received in Farquhar & Gal (2018) for being too simple and unrealistic for CL.

Models Our choices here are largely influenced by previous work (Chaudhry et al., 2019b; Lopez-Paz & Ranzato, 2017). For MNIST & Fashion-MNIST, we train a *multi-layer perceptron* (MLP) consisting of two hidden layers with 250 neurons and ReLU activations. Since we need a model with greater modeling capacity for CIFAR-10 and CIFAR-100, we pick a ResNet-18 (He et al., 2016) pre-trained on ImageNet (Deng et al., 2009).

Hyperparameters We use a learning rate of 0.05 when training the MLP and 0.01 when training the ResNet-18. Both were selected via grid search in the range $[0.1, 0.001]$. Following Aljundi et al. (2019b); Chaudhry et al. (2019b), we set the batch size at $b = 10$, abiding by the assumption that batches provided by the stream should be relatively small, and we perform $n_b = 5$ update steps per incoming batch, as it is a good trade-off between minimizing the training time and maximizing the predictive performance.

3.4. Comparison of Memory Population Methods

Here, we compare the five selected methods (NAIVE, RANDOM, RESERVOIR, GSS and CBRS) over the four selected datasets. Complying with previous work (Aljundi et al., 2019b), we set the memory size at $m = 500$ for MNIST and Fashion-MNIST, while for the more difficult CIFAR-10 and CIFAR-100, we set it at $m = 1000$. We report the 95% confidence interval of the test set accuracy (over ten runs) of each model, after having been trained with the corresponding memory population algorithm. The relevant results are presented in Table 1.

We observe that CBRS outperforms the other four approaches in all four datasets by significant margins. Especially in the case of CIFAR-100 — where some classes are represented only by a single-digit number of instances in the stream — the relative gap over the second-best method is more than 40%. As expected, we notice that the weaker baselines (i.e., NAIVE and RANDOM) are significantly outperformed by the other three approaches.

In all experiments excluding the ablation, all replay methods use weighted replay. By setting up the experiments in this manner, we isolate the difference in performance between the different memory population schemes. In Subsection 3.6, we perform an ablation study that probes the use of the two different types of replay.

Table 1. Comparison of the five learning methods over four different benchmarks. For MNIST and Fashion-MNIST we set the memory size at $m = 500$, while for the two CIFAR- datasets we set it at $m = 1000$. All methods use weighted replay so that they can be compared on equal terms. We report the 95% confidence interval of the accuracy on the test set after the training is complete. Each experiment is repeated for ten different streams.

METHODS	MNIST	F-MNIST	CIFAR-10	CIFAR-100
NAIVE	10.1 ± 0.0	10.0 ± 0.0	10.0 ± 0.0	1.0 ± 0.0
RANDOM	37.8 ± 5.6	38.9 ± 6.7	52.1 ± 3.8	25.8 ± 1.0
RESERVOIR	67.9 ± 3.0	64.1 ± 1.8	54.7 ± 2.2	28.1 ± 1.2
GSS	74.2 ± 3.4	64.9 ± 3.1	63.3 ± 2.3	23.0 ± 1.1
CBRS	$\mathbf{83.3} \pm 2.5$	$\mathbf{75.0} \pm 2.5$	$\mathbf{73.4} \pm 2.2$	$\mathbf{40.2} \pm 1.0$

Table 2. Comparison of the four memory population methods on Fashion-MNIST for various memory sizes. All methods use weighted replay so that they can be compared on equal terms. We report the 95% confidence interval of the accuracy on the test set after the training is complete. Each experiment is repeated for ten different streams.

	MEMORY SIZE m			
METHODS	$m = 100$	$m = 500$	$m = 1000$	$m = 5000$
RANDOM	21.7 ± 4.2	38.7 ± 5.6	57.6 ± 5.1	76.1 ± 1.3
RESERVOIR	53.7 ± 3.0	64.3 ± 2.8	68.2 ± 2.0	75.2 ± 1.9
GSS	59.5 ± 3.6	65.0 ± 3.8	65.1 ± 3.9	70.7 ± 3.2
CBRS	$\mathbf{66.5} \pm 1.5$	$\mathbf{75.8} \pm 2.0$	$\mathbf{76.7} \pm 2.4$	$\mathbf{77.1} \pm 1.8$

3.5. Varying the Memory Size

The goal of this experiment is to survey the impact of memory size in the final learning performance. We evaluate the four methods that use a memory (i.e., all except for NAIVE) over four different memory sizes, ranging from very small (i.e., $m = 100$) to very large (i.e., $m = 5000$). Since we are using only one dataset in this experiment, we opt for one of moderate difficulty, namely Fashion-MNIST. For the same reason mentioned in the previous subsection, all methods use weighted replay. We present the results in Table 2.

The results are consistent across all memory sizes. CBRS outperforms all other methods in every case, with the relative performance improvement being more tangible for smaller memory sizes.

[4]For more details see the supplementary material.

Table 3. Ablation study. We want to isolate the performance boost that CBRS provides both with and without weighted replay. We present results on two different datasets (MNIST and CIFAR-10) and for two different memory sizes ($m = 1000$ and $m = 5000$). We report the 95% confidence interval of the accuracy on the test set after the training is complete. Each experiment is repeated for ten different streams.

	$m = 1000$				$m = 5000$			
	MNIST		CIFAR-10		MNIST		CIFAR-10	
METHODS	UNIFORM	WEIGHTED	UNIFORM	WEIGHTED	UNIFORM	WEIGHTED	UNIFORM	WEIGHTED
RESERVOIR	71.3 ± 2.0	76.3 ± 2.8	54.2 ± 2.1	56.7 ± 1.4	77.9 ± 4.3	86.3 ± 1.8	68.7 ± 4.3	73.1 ± 1.7
GSS	75.4 ± 2.6	77.9 ± 2.3	62.9 ± 2.2	62.8 ± 2.1	76.6 ± 4.8	84.2 ± 1.9	68.5 ± 4.1	71.8 ± 2.0
CBRS	86.0 ± 2.6	86.1 ± 2.6	74.4 ± 2.5	73.7 ± 2.6	85.6 ± 2.8	90.3 ± 1.3	76.8 ± 2.3	78.3 ± 1.9

Table 4. Comparison of the four memory population methods with respect to their computational efficiency. With regard to time complexity, we report the 95% confidence interval of the wall-clock time per incoming batch in milliseconds over ten runs. With regard to memory overhead, we report the relative increase in storage space, compared to storing only the selected data instances, that each method results in. For more details on the contents of this table see Subsection 3.7.

	WALL CLOCK TIME				MEMORY OVERHEAD			
METHODS	MNIST	F-MNIST	CIFAR-10	CIFAR-100	MNIST	F-MNIST	CIFAR-10	CIFAR-100
RANDOM	20.0 ± 0.5	19.3 ± 1.1	363.4 ± 6.4	357.6 ± 4.8	0.00%	0.00%	0.00%	0.00%
RESERVOIR	19.8 ± 0.5	17.7 ± 0.7	367.6 ± 4.6	358.7 ± 4.8	0.00%	0.00%	0.00%	0.00%
GSS	$295.9 \pm 18.$	$316.3 \pm 11.$	$3671.1 \pm 46.$	$3647.8 \pm 27.$	827.19%	827.19%	3639.86%	3639.86%
CBRS	19.3 ± 0.2	20.4 ± 1.0	364.9 ± 5.8	357.9 ± 5.1	0.13%	0.13%	0.03%	0.03%

3.6. Ablation Study

At this point, we would like to isolate the influence of using CBRS in the place of another strong baseline (i.e., RESERVOIR or GSS), from that of using weighted instead of uniform replay. We select an easier dataset (MNIST) and a harder one (CIFAR-10), and measure the accuracy of the trained classifier for all combinations of the three selected memory population methods (i.e., RESERVOIR, GSS, CBRS), two memory sizes ($m = 1000$ and $m = 5000$), and the use of either uniform or weighted replay. Each combination is repeated ten times and the results are presented in Table 3.

There are three points we would like to emphasize here. First, we note that the relative performance gaps between CBRS and the other two methods are higher when using uniform replay. When the memory is populated using RESERVOIR or GSS — two methods that are significantly influenced by the imbalanced distribution of the stream — the probabilities of replaying instances of different classes are very disparate. These disparities, in turn, cause the learner to be able to classify instances of some classes much better than others, and thus deteriorates its accuracy on the test set, where all the classes are more or less balanced.

Second, we observe that using weighted replay is almost always beneficial, compared to using uniform replay. This observation follows naturally from what we described in the previous paragraph. In short, the use of weighted replay partially masks the effects of the imbalanced memory by oversampling the underrepresented classes.

Finally, we notice that when using uniform replay, one of the CBRS entries in Table 3 are worse for $m = 5000$ than they are for $m = 1000$. This abnormality can be intuitively understood by realizing that for the same stream, a memory with higher storage capacity will end up being more imbalanced. In Figure 1 for instance, CBRS would keep the memory perfectly balanced if its size was $m = 255$, but since it actually is $m = 1000$, it cannot.

3.7. Computational Efficiency

An important characteristic of any online CL approach is its computational efficiency. Keeping that in mind, we contrast the time and memory complexity of the four selected replay methods (RANDOM, RESERVOIR, GSS and CBRS).

All experiments are run on an NVIDIA TITAN Xp. For the implementation of GSS, we followed the pseudocode in Aljundi et al. (2019b), while CBRS, RANDOM and RESERVOIR were optimized to the best of our knowledge. The experimental setup is identical to that of Subsection 3.4. All relevant results are presented in Table 4.

Timing the memory population methods is relatively simple. We repeat the learning process for all replay-based methods over ten different streams, and report the 95% confidence interval of the elapsed time per incoming batch in milliseconds. We observe that RANDOM, RESERVOIR and CBRS require roughly the same amount of time. In contrast, GSS is about an order of magnitude more time-consuming than the other three methods. By profiling GSS, we found that its main bottleneck is the additional computation of gradients that are involved in the calculation of the similarity score for each incoming instance.

Quantifying the memory overhead is slightly more complex. First of all, the number we report is a *memory overhead factor*. We define this factor as the additional storage space required by each memory population method, divided by the space that the m stored instances occupy. RANDOM and RESERVOIR require no additional storage space, hence their memory overhead factor is zero in every case. GSS requires an additional storage space for 10 gradient vectors, with each of them containing approximately 3.2×10^5 for the MLP, and 1.1×10^7 for the ResNet-18, 32-bit floating-point numbers. In order for CBRS to be time-efficient, it utilizes a data structure that keeps track of which memory locations correspond to each class, and thus requires an additional storage space equivalent to m 32-bit integers in every case. Therefore, it has a meager 0.13% (MNIST and Fashion-MNIST) and 0.03% (CIFAR-10 and CIFAR-100) memory overhead. Conversely, the additional memory requirements of GSS are equivalent to having an eight (for MNIST and Fashion-MNIST) and 36 (for CIFAR-10 and CIFAR-100) times larger memory, than the one it actually uses for the stored instances.

3.8. Discussion

In this subsection, we attempt to interpret our results qualitatively. As we showed experimentally, CBRS outperforms RESERVOIR and GSS in terms of their predictive accuracy during evaluation. This performance gap can be explained in two steps.

First, a more balanced memory translates into all classes being replayed more or less with the same frequency, and thus not being forgotten. In the opposite case, if a certain class is severely underrepresented in memory it will not be replayed as often, and will thus be more prone to being forgotten. This claim is indirectly exemplified in Table 3, where CBRS is the method that has the least relative increase in accuracy when switching from uniform to weighted replay, since it stores a more balanced subset of the data instances provided by the imbalanced stream.

Second, and more importantly, RESERVOIR and GSS are biased by the imbalanced distribution of the stream and fail to store an adequate number of data instances from highly underrepresented classes, as is demonstrated in Figure 1. As a consequence, these classes are largely forgotten come evaluation time. In contrast, CBRS is guaranteed to store all data instances from significantly underrepresented classes, and thus, is in the best possible position to not forget them in the future. As evidence of this statement, we again refer to Table 3. We observe that even when using weighted replay, CBRS still has a higher accuracy than the other replay methods in all examined cases, as it stores enough instances in order to be able to better remember even the most severely underrepresented classes of the stream.

4. Related Work

There are three main CL paradigms in the relevant research. The first one, called *regularization-based* CL, applies one or more additional loss terms when learning a task, to ensure previously acquired knowledge is not forgotten. Methods that follow this paradigm are, for instance, learning without forgetting (Li & Hoiem, 2017), synaptic intelligence (Zenke et al., 2017) and elastic weight consolidation (Kirkpatrick et al., 2017).

Approaches following the *parameter isolation* paradigm sidestep catastrophic forgetting by allocating non-overlapping sets of model parameters to each task, with relevant examples being the work of Serrà et al. (2018) and Mallya & Lazebnik (2018). Most of such methods require task identifiers during training and prediction time.

Replay-based methods are another major CL paradigm. These methods replay previously observed data instances during future learning in order to mitigate catastrophic forgetting. The replay can take place either directly, via storing a small subset of the observed data (Isele & Cosgun, 2018), or indirectly, with the aid of generative models (Shin et al., 2017).

The majority of research pertinent to CL focuses on the *task-incremental* setting, with some approaches (von Oswald et al., 2020; Shin et al., 2017; Rebuffi et al., 2016) achieving strong predictive performance and at the same time minimizing forgetting.

In contrast, the more challenging *online* CL setting has not been explored as much. Relevant approaches store a small subset of the observed instances, which are then exploited either via replay (Chaudhry et al., 2019b; Aljundi et al., 2019a;b), or by regularizing the training process using data-dependent constraints (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2019a). The latter approach is not applicable to our learning setting since it requires task identifiers both at training and at evaluation time. Therefore, replay-based methods are the only ones that can be practically applied to the minimal-assumption learning setting that we adopt in this work.

Reservoir sampling (Vitter, 1985) extracts an iid subset of the observations it receives, and was, until recently, considered to be the state-of-the-art in selecting which data instances to store during online continual learning (Isele & Cosgun, 2018; Chaudhry et al., 2019b). Aljundi et al. (2019b) introduced two approaches (i.e., GSS-IQP and GSS-Greedy) that try to maximize the variance of the stored memories with respect to the gradient direction of the model update they would generate. GSS-IQP utilizes *integer quadratic programming*, while GSS-Greedy is a more efficient heuristic approach that actually outperforms GSS-IQP. Additionally, Aljundi et al. (2019b) demonstrate that both their proposed algorithms achieve higher accuracy than reservoir sampling when learning moderately imbalanced streams of MNIST (LeCun et al., 2010) digits with two classes per task.

5. Conclusion

In this work, we examined the issue of online continual learning from severely imbalanced, temporally correlated streams. Moreover, we proposed CBRS — an efficient memory population approach that outperforms the current state of the art in such learning settings. We provided an interpretation for the performance gap between CBRS and the current state of the art, accompanied by supportive empirical evidence. Specifically, we argued that underrepresented classes in memory tend to be forgotten more quickly, partly because they are not replayed as often as more sizeable classes, and also because of the scarcity of their corresponding stored instances. Further improvements in replay-based CL methods could be possible, if memory population methods are able to infer which classes are more challenging to learn and store their instances preferentially.

Acknowledgements

We would like to thank Ivan Vulić and Pieter Delobelle for providing valuable feedback on this work. In addition, we want to thank the anonymous reviewers for their comments. This work is part of the CALCULUS[5] project, which is funded by the ERC Advanced Grant H2020-ERC-2017-ADG 788506.

[5]http://calculus-project.eu

References

Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., and Page-Caccia, L. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems 32*, pp. 11849–11860. 2019a.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems 32*, pp. 11816–11825. 2019b.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Branco, P., Torgo, L., and Ribeiro, R. P. A survey of predictive modeling on imbalanced domains. *ACM Comput. Surv.*, 49(2), August 2016.

Cangelosi, A. and Schlesinger, M. *Developmental Robotics: From Babies to Robots*. MIT Press, 2015.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*, 2019a.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486, 2019*, 2019b.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383v1*, 2019.

Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Ego-Stengel, V. and Wilson, M. A. Disruption of ripple-associated hippocampal activity during rest impairs spatial learning in the rat. *Hippocampus*, 20(1):1–10, 2010.

Farquhar, S. and Gal, Y. Towards Robust Evaluations of Continual Learning. In *Lifelong Learning: A Reinforcement Learning Approach workshop, ICML*, 2018.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128 – 135, 1999.

Girardeau, G., Benchenane, K., Wiener, S. I., Buzsáki, G., and Zugaro, M. B. Selective suppression of hippocampal ripples impairs spatial memory. *Nature Neuroscience*, 12 (10):1222, 2009.

Goodfellow, I. J., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Isele, D. and Cosgun, A. Selective experience replay for lifelong learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. 2012.

LeCun, Y., Cortes, C., and Burges, C. MNIST handwritten digit database. *Yann LeCun's Website*, 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

Lopez-Paz, D. and Ranzato, M. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 30*, pp. 6467–6476. 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation; Volume 24*, pp. 109 – 165. 1989.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Rebuffi, S.-A., Kolesnikov, A. I., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5533–5542, 2016.

Serrà, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems 30*, pp. 2990–2999. 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, 2018.

Tani, J. *Exploring Robotic Minds: Actions, Symbols, and Consciousness as Self-Organizing Dynamic Phenomena*. Oxford University Press, 2016.

van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Vitter, J. S. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, March 1985.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3987–3995, 2017.