Online Continual Learning from Imbalanced Data with Kullback-Leibler-loss based replay buffer updates

Sotirios Nikoloutsopoulos[1*] **Iordanis Koutsopoulos**[1] **Michalis K. Titsias**[2]
[1]Athens University of Economics and Business
[2]DeepMind
{snikolou,jordan}@aueb.gr
mtitsias@google.com

Abstract

We propose an online replay-based Continual Learning policy, in which the learner stores data points to a local buffer and replays it during training. The core of our contribution is a new replay buffer contents update policy that combines a Kullback-Leibler (K-L) loss and an appropriate modification of the celebrated Reservoir Sampling algorithm. The decisions at each time are, whether the newly arriving training data points will be inserted in the buffer, and which existing data points from the buffer will be substituted. We update the buffer content so that the proportion of stored data points from different classes in the buffer approximates a target distribution that depends on the empirical distribution of classes seen in the training data stream. We parameterize the target distribution with a single parameter that allows us to model different target class distributions in the buffer, such as the class distribution that is present in the training data stream, the uniform class distribution, and a distribution with class percentages that are inversely proportional to those in the training data stream. We evaluate our method on MNIST, Fashion-MNIST, CIFAR-10 and CIFAR-100, and we show that our method is superior to the state-of-the-art Reservoir Sampling algorithm. Our main finding is that the best (in terms of accuracy and forgetting) value of the parameter that determines the distribution of classes in the buffer versus that of the stream depends on statistics of the training data and on the dataset itself. Our work paves the way for further work to learn this parameter in the realistic scenario that it is unknown, thus contributing to the objective of an optimal replay-based continual learning approach that adapts to the specifics of each scenario.

1 Introduction

In online supervised continual learning, the learner receives a batch of labelled training data at each time slot, and its goal is to train at each time slot a Machine Learning (ML) model that would perform well in the inference (test) tasks in that time slot. Due to memory constraints, the learner cannot maintain all previously seen training data and use it to train a model; instead, it can only maintain a small portion of it. An important issue in this setting is the non-stationary distribution of training data which manifests itself as a time-varying empirical distribution of the classes seen in the training examples over time. Therefore, classes may appear with an unpredictable proportion in the training data, and their distribution may be inherently imbalanced in the long run.

Since the statistics of the test data stream are often non-stationary as well, the major challenge in Continual Learning is to satisfy any arising inference requests, by maintaining at each time an

*Corresponding author

Accepted for Workshop on Human in the Loop Learning (NeurIPS HILL 2022).

informed model that has not forgotten acquired knowledge from past seen training data [12], and at the same time it incorporates new knowledge from the new training data batch. Incorporating new knowledge that hurts performance on past inference tasks is known as catastrophic forgetting [12]. Approaches in the literature that prevent catastrophic forgetting can be roughly categorized as follows: (*i*) regularization methods that penalize changes in neural network (NN) weights that are important to previous seen tasks [8, 1, 14, 21, 10]; (*ii*) Dynamic expandable NN architectures that grow the number of NN weights to fit new knowledge [13, 14, 20]; (*iii*) knowledge distillation [3, 19, 9]; and (*iv*) replay-based methods [11, 16, 17], which operate with a replay buffer, in which a small portion of selected past training data are retained and used together with the current training data to update the model. Our approach falls within the class of replay-based methods.

Previous replay-based approaches in the literature have focused on different goals. For example, the Reservoir Sampling (RS) algorithm keeps in the buffer a subset of data points that are representative of all training data that are seen up to that time. In that sense, the proportion of classes in the buffer tries to approximate the (time-varying) distribution of classes in the training data stream. On the other hand, the work [4] aims to balance as much as possible the stored data points with respect to their class labels.

The performance of these methods in terms of test accuracy is very much dependent on the class distribution of the training and the test data streams. Because the methods above take into account only the class distribution in the training data, they perform well only when the class distributions in the training data and in the test request stream are similar. However, these two distributions are not necessarily similar. This phenomenon is exacerbated in online settings where non-stationarity appears in both the training data and the test request data distribution.

For example, if the class distribution is imbalanced in the training data and balanced in the test requests stream, the CBRS algorithm in [4] tries to store in the buffer data points so that the (imbalanced) class distribution in the stream is maintained. If classes are requested for inference with almost equal probability, the classes that are underrepresented in the buffer will exhibit poor test accuracy. In another example, if the class distribution is balanced in the training data but imbalanced in the test requests stream, both the RS and the CBRS algorithm will store approximately the same number of data points from each class in the buffer. However, this is not fair for classes that are requested with higher probability for inference, in the sense that the most frequently requested classes should be given some type of priority. Also, in this case, the buffer space is unnecessarily consumed for classes that are rarely or never requested in the test stream; the same space could be used for data points of classes that are actually requested.

It becomes apparent that the appropriate proportion of classes that should be maintained in the buffer highly depends on the unknown class distribution in the training data stream, on the dynamics of training data and the inference task streams, and also on the nature of the training dataset itself. In this work, we set off to design a replay-based continual learning algorithm that can decide to populate the buffer with a range of target class distributions which are determined by the training data stream. We propose a replay buffer content update policy that combines a Kullback-Leibler (K-L) loss and an appropriate modification of the celebrated Reservoir Sampling algorithm. The decisions at each time slot are, whether the newly arriving training data points will be inserted in the buffer, and which existing data points from the buffer will be substituted. Specifically, we update the buffer content so that the proportion of stored data points from different classes in the buffer approximates a target distribution that depends on the empirical distribution of classes seen in the training data stream.

We parameterize this target distribution with a single parameter that allows us to model different target class distributions in the buffer, ranging from the class distribution in the training data stream, the uniform class distribution, and a distribution with class percentages that are inversely proportional to those in the training data stream.

The first contribution of our work is that our method is equivalent to CBRS [4] when the target distribution is equal to the uniform class distribution. In that case, the decision making process stores approximately the same number of data points per class in the buffer. The second contribution, which we empirically see from our experiments is that we can find at least one target distribution that is not equal to the uniform distribution, and that target distribution achieves higher test accuracy from CBRS [4]. These findings point to the direction of further exploring the problem of identifying the best target distribution, i.e. computing the best parameter value, and accordingly perform buffer updates.

2 Proposed approach

2.1 Continual Learning setting and notation

We consider a supervised Continual Learning setting with a learner and an infinite stream of training data. At each time step t, the learner receives a batch of training data $\mathcal{B}_t = \{(x_{j,t}, y_{j,t})\}_{j=1}^{N}$ from the stream, where $\{x_{j,t}\}_{j=1}^{N}$ are the input data samples (e.g. images) at time t, and $\{y_{j,t}\}_{j=1}^{N}$ are the corresponding target outputs (e.g. class labels). The learner has a buffer with a set of contents denoted as set $\mathcal{M}$ and fixed capacity M. We assume that, upon receiving the new data batch $\mathcal{B}_t$, the learner performs a number of update steps over the new batch and over a random batch $\mathcal{R}_t$ chosen (replayed) from the buffer. We assume a Deep Neural Network (DNN) model that consists of a number of layers with overall parameter vector θ. The number of outputs of the final layer, i.e. the size of the feature vector, is K which is equal to the number of class labels. The model receives an input x and constructs in its final output a representation or feature vector $\phi(x;\theta) \in \mathbb{R}^K$. We map the feature vector to classification probabilities with $f(x;\theta) = \text{softmax}(\phi(x;\theta))$

The loss function for an arbitrary batch $\mathcal{D}$ of size N, where $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, is

$$L_{\mathcal{D}} = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i) \tag{1}$$

The purpose of training in continual learning is to optimize the model parameters θ_t at each time slot t, with a loss function $\ell(\cdot)$ Let $L_{\mathcal{B}_t}$, and $L_{\mathcal{R}_t}$ be the loss over $\mathcal{B}_t$ and $\mathcal{R}_t$ respectively. We define the total loss at time t as

$$L_t = \beta L_{\mathcal{B}_t} + (1-\beta) L_{\mathcal{R}_t} \tag{2}$$

, where β is a parameter that trades off between the loss over data in the current batch and that over buffer data.

Let $\mathbf{n}_t = (n_{t,1}, \ldots, n_{t,K})$ be the vector whose i-th component $n_{t,i}$ is a counter that shows the number of seen data points of class i up to time t from the training data stream. Let $s_{t,i} = n_{t,i} / \sum_{k=1}^{K} n_{t,k}$ be the proportion of data points of class i in the training data up to time t. We denote the empirical distribution of classes up to time t by vector $\mathbf{s}_t = (s_{t,1}, \ldots, s_{t,K})$.

For the buffer, we define vector $\mathbf{m}_t = (m_{t,1}, \ldots, m_{t,K})$ whose i-th component $m_{t,i}$ is the number of stored data points of class i in the buffer at time t, with $\sum_{k=1}^{K} m_{t,k} = M$. Let $\mathcal{C}_t$ be the set of classes that are present in the buffer at time t, i.e. class $k \in \mathcal{C}_t$ if and only if $m_{t,k} \geq 1$.

2.2 Decision making for buffer update

For the first M time steps of the process, the agent stores incoming data points until the buffer is full. For each time $t > M$, the process goes as follows. After the learner finishes training on batch $\mathcal{B}_t$, it must decide which of the incoming data points (x_*, y_*) in $\mathcal{B}_t$ must be inserted into the buffer, and which data points from the buffer need to be ejected. The learner has the following options: (i) insert (x_*, y_*) into the buffer and remove from the buffer a data point from the same class y_*, (ii) insert (x_*, y_*) in the buffer and replace a data point from a class other than y_*, (iii) do nothing and ignore the new data point.

We define parameter $a \in \mathcal{R}$, and we construct a target class distribution which depends on the training data stream class distribution, $\mathbf{p}_t = \mathbf{s}_t^a$, where $\mathbf{p}_t$ is the vector whose i-th component is $p_{i,t} = s_{i,t}^a$. Observe that this parameterized target class distribution can model different types of class distribution:

- If $a = 0$, then $\mathbf{p}_t$ reduces to $(\frac{1}{K}, \ldots, \frac{1}{K})$, the uniform class distribution. In this case, we do not take into account the class distribution in the training data stream, and our method is equivalent to CBRS which strives for an (almost) equal share of buffer capacity for each class. See Figure 1 for a pictorial demonstration on an hypothetical imbalanced stream.
- If $a = 1$, then $\mathbf{p}_t = \mathbf{s}_t$, the class distribution in the stream. In this case, we store data points in the buffer according so as to resemble distribution in the training data stream.

- If $a = -1$, then $p_{i,t} \propto 1/s_{i,t}$, the percentage of data points of a specific class in the buffer is inversely proportional to the class percentage seen in the training data stream distribution. This means that we give emphasis to maintaining in the buffer data points from classes that are not often encountered in the training data stream.

For each data point (x_*, y_*) in the training data batch $\mathcal{B}_t$ at time t, the learner proceeds as follows. For each class $k \in \mathcal{C}_t$, the learner tentatively removes from the buffer a randomly selected data point of class k and substitutes it with data point (x_*, y_*). For each class $k \in \mathcal{C}_t$, this tentative operation forms a new vector of counts for the buffer

$$\mathbf{m}^k = \mathbf{m}_t - \mathbf{e}_k + \mathbf{e}_{y_*} \tag{3}$$

where $\mathbf{e}_k$ denotes the binary vector of K components, whose k-th component is 1 and all others are zero.

Now, we define vector $\hat{\mathbf{m}}_t^k = \frac{1}{M}\mathbf{m}_t^k$, whose i-th component shows the proportion of data points of class k in the buffer *after* the tentative inclusion of the new data point and removal of a data point of class k. Next, the learner computes the following Kullback-Leibler distance:

$$V_t^k = \mathrm{KL}(\mathbf{p}_t || \hat{\mathbf{m}}_\mathbf{t}{}^k) \tag{4}$$

that represents the distance of class vector $\hat{\mathbf{m}}_\mathbf{t}$ from the empirical target class distribution $\mathbf{p}_t$ which in turn depends on the class distribution seen in the stream up to time t.

Then, the learner identifies the class with the minimum tentative KL distance from $\mathbf{p}_t$,

$$k_* = \arg \min_{k \in \mathcal{C}_t \cup \{y_*\}} V_t^k. \tag{5}$$

Algorithm 1 KLRS - Kullback–Leibler Reservoir Sampling

Input: $\mathcal{M}$ is the set of buffer contents (data points); (x_*, y_*) is the incoming data point from batch $\mathcal{B}_t$ that we examine; $\mathbf{m}_t$ is the vector of counters of data points per class in the buffer; $\mathbf{s}_t$ is the vector of counters per class in the training data stream; $\mathbf{p}_t$ is the vector of target probabilities.

Buffer_Update(x_*, y_*)**:**

```
if |M| < M then                                   ▷ i.e. buffer is not full
    M ← M ∪ (x_*, y_*)                            ▷ Store (x_*, y_*) in the buffer
else
    for each class k in C_t do
        m^k = m_t − e_k + e_{y_*}                 ▷ Remove a data point from class k and substitute it with the data point (x_*, y_*).
        m̂_t^k = (1/M) m_t^k                       ▷ The distribution of classes in the buffer after the substitution
        V_t^k = KL(p_t || m̂_t^k)                  ▷ Utility between the target distribution and the tentative memory distribution
    end for
    k_* = arg min_{k ∈ C_t ∪ {y_*}} V_t^k         ▷ Find class with minimum utility
    if k_* = y_* then                             ▷ If class y_* has the minimum utility
        Generate a random number r ∈ [1, n_{t,k^*}]   ▷ Perform Class-based Reservoir
        if r ≤ m_{t,y_*} then
            Insert (x_*, y_*) in the buffer in place of another data point of the same class
        else
            Ignore (x_*, y_*)
        end if
    else                                          ▷ Choose one data point with minimum utility at random
        Choose a random data point from class k^* from the buffer and substitute it with (x_*, y_*).
    end if
end if
```

The final step is split into two cases.

- if $k_* = y_*$, namely if the class that achieves the smallest KL distance is the same as the one of (x_*, y_*), the agent performs a class-based Reservoir Sampling. Namely the agent generates a random number $r_t \in [1, n_{t,k_*}$. If $r_t \leq m_{k_*,t}$, the learner inserts (x_*, y_*) in the buffer in place of another data point of the same class; otherwise the agent ignores the data point.

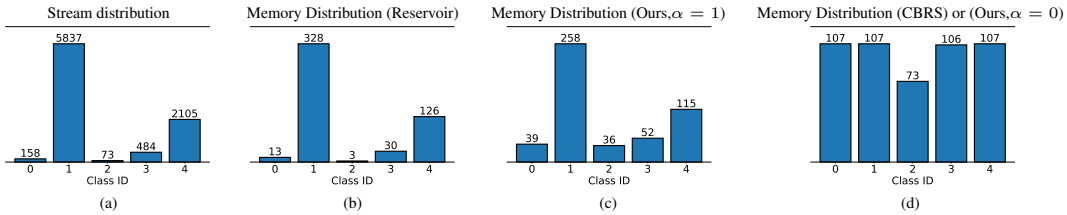

Figure 1: Hypothetical Stream population (a), Reservoir Sampling (b), Our method with $\alpha = 1$ (c), and CBRS and our method with $\alpha = 0$ (d). In this hypotherical example the capacity of the buffer is 500.

- if $k_* \neq y_t$, the learner chooses a random data point from class k_* from the buffer and substitutes it with (x_*, y_*).

Algorithm 1 summarizes the steps described above for the decision making process of our proposed method, which we call Kullback–Leibler Reservoir Sampling (KLRS). The proposed KLRS algorithm is part of the general continual learning (CL), and constitutes the portion of the CL algorithm that relates to buffer update. The general Continual Learning process involves the following stages for each data batch: (*i*) train the model with the current batch and a random batch drawn from the buffer; (*ii*) run the KLRS algorithm. The general CL process is summarized under Algorithm 2.

Algorithm 2 The Continual Learning Process

Input: θ is the model parameter; $\mathcal{B}_t$ is the batch of training data points from the stream; $\mathcal{R}_t$ is the random batch drawn from the buffer; η is the learning rate; β is the relative importance weight between the training data batch loss and the buffer data batch loss; $|\mathcal{C}_t|$ is the number of classes seen so far in the stream.

While stream has next batch:

$\mathcal{B}_t$ = (next batch from stream)

$$\beta = \begin{cases} 1, & |M| = 0 \\ \frac{1}{|C_t|}, & \text{otherwise} \end{cases}$$

for $i = 1..\tau$ **do** ▷ Training process

$\mathcal{R}_t \sim \mathcal{M}$ ▷ Sample without replacement the replay batch from the buffer

$L_t = \beta L_{\mathcal{B}_t} + (1 - \beta) L_{\mathcal{R}_t}$ ▷ Compute joint loss

$\nabla_\theta L_t = \beta \nabla_\theta L_{\mathcal{B}_t} + (1 - \beta) \nabla \theta L_{\mathcal{R}_t}$ ▷ Compute joint gradient

$\theta \leftarrow \theta - \eta \nabla_\theta L_t$ ▷ Perform gradient update on the model parameter

end for

for each data point (x_*, y_*) in $\mathcal{B}_t$ **do** ▷ Populate buffer

Buffer_Update(x_*, y_*) ▷ Try to insert data point (x_*, y_*) into the buffer

end for

3 Experiments

3.1 Experimental Setup

We repeat each experiment 5 times with 5 different imbalanced streams. We assume a class-incremental setup, and that there is an arbitrary class ordering among the classes. The training data in the stream appears with respect to the class ordering [4]. We clarify that the class ordering of the classes is arbitrary but consistent for each experiment. For more details about the training stream counts and the test set counts per label, please see Appendix A.

We evaluate the efficiency of each method under two test sets: (i) the "original test set" where the test set contains all the data from the initial test set, (ii) the "imbalanced test set" where the class components are proportional to the empirical distribution of the classes in the training data. We collect the Test Accuracy, and the Forgetting metric that occurred at the last time step t, from the 5 independent experiments. Then, from these collected values, we report the 95% confidence interval of the Average Test Accuracy, and of the Average Forgetting metric.

Furthermore, we consider two replay schemes at the buffer:

- Uniform replay, in which all the data points in the buffer have equal probability of being replayed. However, a class with few data points in the buffer will have a low probability of being replayed.
- Weighted replay, where the idea is to give higher importance to the classes with the fewest examples. That is, weighted replay favors under-represented classes in the buffer, so that these data from the buffer are replayed more often, because we want to prevent catastrophic forgetting for these classes. At time step t, we define the weighted probability $pr(m_{t,i})$ for selecting a data point of class i from the buffer. We reverse the importance per class i by raising each count $m_{t,i}$ to a negative number e.g. -1, thus:

$$pr(m_{t,i}) = \frac{m_{t,i}^{-1}}{\sum_{j \in \mathcal{C}_t} m_{t,j}^{-1}} \tag{6}$$

The baselines we consider for comparison are: Reservoir Sampling [18] and CBRS [4].

- RS: The Reservoir Sampling (RS) method stores a subset of data points from a stream of unknown infinite length. The key concept of the Reservoir Sampling is that each data point in the stream has equal probability to be stored in the buffer.
- Class-Balanced RS (CBRS): An extension of the RS approach, in which the learner tries to preserve class balance within the buffer. When balanced is achieved for a class, this algorithm applies the concept of the RS for that specific class.

3.1.1 Datasets, Hyperparameter search, and Setup for class imbalance

We evaluate our KLRS method and the baselines [4, 18] on the MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100 datasets. For all methods we set the sizes of both the training batch $\mathcal{B}_t$ and of the replay batch $\mathcal{R}_t$ equal to 10. We set the learning rate for MNIST and Fashion-MNSIT to 0.05 [4]. For CIFAR-10 and CIFAR-100, we start from a pre-trained Resnet18 neural network architecture [6] on the ImageNet dataset [5], and we set the learning rate to 0.01 [4]. We perform an hyperparameter search for α in the set of values $\pm\{1.0, 0.75, 0.5, 0.25, 0.2, 0.15, 0.1, 0.05\}$

We apply the same imbalanced setup per label in the training stream as in [4], in which they start from a set of 5 imbalanced factors $\mathbf{r} = \{10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}\}$. For K classes, we define the imbalanced factors $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_K)$, where $\epsilon_j \in \mathbf{r}$, $j = 1 \ldots K$, and ϵ_j is the imbalanced factor for class j. If $K \leq 5$, then $\boldsymbol{\epsilon}$ is set to a random subset of $\mathbf{r}$ with K components without replacement. In our experiments $K > 5$, we repeatedly choose 5 classes from the K classes that have not been assigned an imbalanced factor yet, and then we randomly distribute the factors from $\mathbf{r}$ to those 5 classes without replacement.

3.1.2 CL Perfomance Metrics

We measure the performance of the learner with some metrics at a given time step t that the learner receives a new batch of data from the stream. Note that these metrics are not part of the decision-making process. We define as a_t^i the average test-accuracy of class i at time step t. Then, the average test accuracy for all K classes at time step t is

$$A_t = \frac{1}{K} \sum_{c=1}^{K} a_t^c \tag{7}$$

Another metric used in Continual Learning is the Forgetting metric, which measures how much the accuracy performance of the model reduces with time, therefore it measures the extent to which the model forgets after it is trained with the batch from the stream. The forgetting metric at time step t, is defined similarly to [15] in task-free settings,

$$F_t = \frac{1}{K} \sum_{c=1}^{K} \max_j (a_j^c - a_t^c) \tag{8}$$

where $j = 1, \ldots, t-1$.

3.2 Results and discussion

Table 1: Test Accuracy for different values of capacity M, on the MNIST and Fashion-MNIST datasets, when the test inference is the original test set.

	M=1000				M=2000			
	MNIST		Fashion-MNIST		MNIST		Fashion-MNIST	
Methods	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted
RS	69.62 ± 1.67	74.36 ± 0.61	63.26 ± 3.14	68.20 ± 2.65	77.23 ± 2.03	82.35 ± 1.35	64.50 ± 2.85	72.92 ± 1.68
CBRS	88.78 ± 1.01	88.86 ± 1.07	78.27 ± 1.07	79.61 ± 0.32	90.37 ± 1.25	90.51 ± 1.10	76.93 ± 3.35	80.24 ± 0.97
KLRS	$\mathbf{89.57 \pm 0.99}$	$\mathbf{89.36 \pm 0.98}$	$\mathbf{80.47 \pm 0.62}$	$\mathbf{80.27 \pm 0.29}$	$\mathbf{90.47 \pm 1.45}$	$\mathbf{90.64 \pm 1.08}$	$\mathbf{81.05 \pm 0.88}$	$\mathbf{81.27 \pm 0.80}$

Table 2: Forgetting for different values of capacity M, on the MNIST and Fashion-MNIST datasets, when the test inference is the original test set.

	M=1000				M=2000			
	MNIST		Fashion-MNIST		MNIST		Fashion-MNIST	
Methods	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted
RS	26.46 ± 2.24	22.84 ± 0.56	31.72 ± 2.84	29.48 ± 2.51	18.94 ± 2.37	15.11 ± 1.45	30.87 ± 2.21	24.66 ± 1.67
CBRS	8.73 ± 1.54	8.97 ± 1.26	19.95 ± 0.96	18.75 ± 0.42	7.00 ± 1.34	7.32 ± 1.22	20.83 ± 3.16	18.19 ± 1.02
KLRS	$\mathbf{8.10 \pm 1.07}$	$\mathbf{8.50 \pm 1.20}$	$\mathbf{17.73 \pm 1.14}$	$\mathbf{18.16 \pm 0.46}$	$\mathbf{6.88 \pm 1.66}$	$\mathbf{7.21 \pm 1.28}$	$\mathbf{16.61 \pm 1.23}$	$\mathbf{17.24 \pm 0.74}$

Table 3: Test Accuracy for different values of capacity M, on the MNIST and Fashion-MNIST datasets, when the test inference is the imbalanced test set.

	M=1000				M=2000			
	MNIST		Fashion-MNIST		MNIST		Fashion-MNIST	
Methods	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted
RS	70.14 ± 1.23	76.23 ± 1.14	63.25 ± 2.16	68.29 ± 1.20	79.03 ± 2.73	83.18 ± 2.18	63.51 ± 4.20	71.70 ± 2.64
CBRS	89.15 ± 1.84	88.74 ± 1.43	77.77 ± 2.03	79.57 ± 0.81	90.78 ± 0.64	90.41 ± 0.90	76.82 ± 3.39	80.74 ± 1.20
KLRS	$\mathbf{89.76 \pm 1.14}$	$\mathbf{89.99 \pm 1.21}$	$\mathbf{79.81 \pm 1.52}$	$\mathbf{80.63 \pm 1.17}$	$\mathbf{91.19 \pm 1.51}$	$\mathbf{91.34 \pm 0.99}$	$\mathbf{81.53 \pm 0.94}$	$\mathbf{81.43 \pm 0.88}$

Table 4: Forgetting for different values of capacity M, on the MNIST and Fashion-MNIST datasets, when the test inference is the imbalanced test set.

	M=1000				M=2000			
	MNIST		Fashion-MNIST		MNIST		Fashion-MNIST	
Methods	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted
RS	25.33 ± 2.79	20.77 ± 2.07	32.48 ± 1.51	29.83 ± 2.05	17.96 ± 3.10	14.35 ± 2.96	32.38 ± 3.24	26.57 ± 2.79
CBRS	9.10 ± 2.35	9.63 ± 1.74	21.40 ± 2.14	19.21 ± 0.97	7.37 ± 1.25	8.14 ± 1.19	21.80 ± 3.33	18.48 ± 1.02
KLRS	$\mathbf{8.63 \pm 1.17}$	$\mathbf{8.38 \pm 1.09}$	$\mathbf{19.32 \pm 1.13}$	$\mathbf{18.40 \pm 0.96}$	$\mathbf{6.69 \pm 1.67}$	$\mathbf{7.27 \pm 1.30}$	$\mathbf{17.00 \pm 1.17}$	$\mathbf{17.75 \pm 0.72}$

Table 5: Test Accuracy for the CIFAR-10 and the CIFAR-100 dataset with capacity $M = 2000$, when the test inference is the imbalanced test set.

	CIFAR-10		CIFAR-100	
Methods	Uniform	Weighted	Uniform	Weighted
RS	54.49 ± 2.24	55.92 ± 3.02	27.11 ± 0.8	29.43 ± 0.82
CBRS	69.52 ± 4.29	72.19 ± 3.64	35.58 ± 1.46	35.94 ± 1.59
KLRS	$\mathbf{72.82 \pm 3.71}$	$\mathbf{72.24 \pm 3.01}$	$\mathbf{37.11 \pm 0.79}$	$\mathbf{36.59 \pm 1.64}$

For MNIST and Fashion-MNIST, we report the test accuracy, and the forgetting, for buffer size $M = 1,000$ and $M = 2,000$. At Tables 1, 2 we report the statistics for the original test set, and at Tables 3,4 we report the statistics for the imbalanced test set . For CIFAR-10 and CIFAR-100 we report the test accuracy for the imbalanced test set at Table 5 with $M = 2000$. We can clearly see that our method outperforms RS [18], and CBRS [4], as it achieves the highest average Test Accuracy at each dataset. Furthermore, we verify that the corresponding average Forgetting metric is lower than that of RS [18], and CBRS [4]. Also, we observe that the performance of RS [18], and CBRS [4] degrades as we increase the size of the buffer at the Fashion-MNIST dataset with the uniform replay scheme, but our method performs even better when we increase the buffer size. Lastly, for each setting we provide the value of α that achieved the highest test accuracy at the Appendix A.

4 Conclusion

We propose a replay-based continual learning algorithm that populates the buffer so as to attain different target class distributions in it, ranging from the class distribution in the training data stream, the uniform class distribution, and a distribution with class percentages that are inversely proportional to those in the training data stream. Our method, termed Kullback-Leibler Reservoir Sampling (KLRS) combines a Kullback-Leibler (K-L) distance metric and a modification of the celebrated Reservoir Sampling algorithm and decides at each time slot how to populate the buffer so that the proportion of data points from different classes in the buffer tracks the target class distribution that depends on the time-varying distribution of classes seen in the training data stream.

Our main finding through data experiments is that the best (in terms of accuracy and forgetting) value of the parameter that determines the distribution of classes in the buffer versus that of the stream depends on statistics of the training data and on the dataset itself. In this work, we assumed different, but fixed values of this parameters. A future step in the case that this parameter is unknown would be to attempt to learn this parameter on the fly.

Code

We provide our code at https://github.com/sotirisnik/KLRS. The implementation is in Haiku [7], which is based on the Jax [2] library. We also use the Pytorch library only to transfer the pre-trained Resnet18 [6] weights from Imagenet[5] to Haiku.

Acknowledgments

This work was supported by the CHIST-ERA grant CHIST-ERA-18-SDCDN-004 (project LeadingEdge, grant number T11EPA4- 00056) through the General Secretariat for Research and Innovation (GSRI).

References

[1] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11246–11255, 2019.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 241–257, Cham, 2018. Springer International Publishing.

[4] Aristotelis Chrysakis and Marie-Francine Moens. Online continual learning from imbalanced data. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1952–1961. PMLR, 13–18 Jul 2020.

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

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[7] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[8] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, mar 2017.

[9] Kibok Lee, Kimin Lee, Jinwoo Shin, and Honglak Lee. Overcoming catastrophic forgetting with unlabeled data in the wild. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 312–321, 2019.

[10] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

[11] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6470–6479, Red Hook, NY, USA, 2017. Curran Associates Inc.

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

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

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

[15] Murray Shanahan, Christos Kaplanis, and Jovana Mitrović. Encoders and ensembles for task-free continual learning, 2021.

[16] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 2994–3003, Red Hook, NY, USA, 2017. Curran Associates Inc.

[17] Gido M. van de Ven and Andreas Savas Tolias. Generative replay with feedback connections as a general strategy for continual learning. *ArXiv*, abs/1809.10635, 2018.

[18] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, mar 1985.

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

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

[21] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3987–3995. JMLR.org, 2017.

A Counts per label and Best values for α

In this section, we provide the counts per label $i = 0 \dots 9$, for the 5 imbalanced training streams we use in our experiments, and the counts for their corresponding test sets. See Table 6 for MNIST, and Table 7 for Fashion-MNIST. For each dataset, we start from its original counts, and we keep 90% of the population per label i(see Initial Stream row). After that, we construct each of the 5 imbalanced training streams by distributing the imbalanced factors per label i. Note that MNIST is the only dataset whose original counts per label i are not balanced. Lastly, we provide the best values for the parameter α at Table 8 for MNIST and Fashion-MNIST. For CIFAR-10 the best value of α for the uniform replay is -0.5, while for the weighted replay is 0.05. For CIFAR-100 we set $\alpha = -0.25$. For the single parameter α we observe that

- When we use the uniform replay, the best value for the hyperparameter α is negative.
- At the Fashion-MNIST dataset, the best value for α is negative.
- At the MNIST dataset, the best value for α is positive, except from the case in which the buffer size is 1000.
- The hyperparameter search can be reduced to the range $[-0.5, 0.5]$.

Table 6: Counts per label $i = 0 \dots 9$ for the initial training stream(90% of the original population in the training set per class i), and the test set on the MNIST dataset.

	Training Stream									
Counts/Labels	0	1	2	3	4	5	6	7	8	9
Initial Stream	5330	6067	5362	5517	5257	4878	5326	5638	5265	5354
1st Imbalanced Stream	533	1918	53	5517	525	1542	5326	178	52	169
2nd Imbalanced Stream	5330	1918	169	551	5257	48	532	56	166	1693
3rd Imbalanced Stream	1685	60	536	174	1662	154	5326	56	5265	535
4th Imbalanced Stream	1685	6067	169	55	1662	4878	168	563	52	535
5th Imbalanced Stream	533	1918	169	55	166	487	5326	56	5265	1693
	Test Set									
Original Test Set	980	1135	1032	1010	982	892	958	1028	974	1009
1st Imbalanced Test Set	98	358	10	1010	98	282	958	32	9	31
2nd Imbalanced Test Set	980	358	32	101	982	8	95	10	30	319
3rd Imbalanced Test Set	309	11	103	31	310	28	958	10	974	100
4th Imbalanced Test Set	309	1135	32	10	310	892	30	102	9	100
5th Imbalanced Test Set	98	358	32	10	31	89	958	10	974	319

Table 7: Counts per label $i = 0..9$ for the training stream, and the test set on the Fashion-MNIST dataset.

	Training Stream									
Counts/Labels	0	1	2	3	4	5	6	7	8	9
Initial Stream	5400	5400	5400	5400	5400	5400	5400	5400	5400	5400
1st Imbalanced Stream	540	1707	54	5400	540	1707	5400	170	54	170
2nd Imbalanced Stream	5400	1707	170	540	5400	54	540	54	170	1707
3rd Imbalanced Stream	1707	54	540	170	1707	170	5400	54	5400	540
4th Imbalanced Stream	1707	5400	170	54	1707	5400	170	540	54	540
5th Imbalanced Stream	540	1707	170	54	170	540	5400	54	5400	1707
	Test Set									
Initial Test Set	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
1st Imbalanced Test set	100	316	10	1000	100	316	1000	31	10	31
2nd Imbalanced Test Set	1000	316	31	100	1000	10	100	10	31	316
3rd Imbalanced Test Set	316	10	100	31	316	31	1000	10	1000	100
4th Imbalanced Test Set	316	1000	31	10	316	1000	31	100	10	100
5th Imbalanced Test Set	100	316	31	10	31	100	1000	10	1000	316

Table 8: Best α values for KLRS for different values of capacity M, on the MNIST and Fashion-MNIST datasets.

	M=1000				M=2000			
	MNIST		Fashion-MNIST		MNIST		Fashion-MNIST	
Test set	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted	Uniform	Weighted
Initial	-0.05	0.05	-0.25	-0.25	-0.2	0.05	-0.25	-0.5
Imbalanced	-0.05	-0.1	-0.2	-0.25	-0.2	0.1	-0.25	-0.15

B Kullback-Leibler loss

Let $\mathbf{p}$ and $\mathbf{q}$ be vectors of length K, whose components p_i, and q_i is the probability per label $i = 1 \dots K$. A loss function that computes the difference or deviation from the probability vector $\mathbf{q}$ to $\mathbf{p}$ is the KL loss, also known as relative entropy, which *measures* how much information is lost when we move from the vector $\mathbf{q}$ to the vector $\mathbf{p}$. The KL loss function from $\mathbf{q}$ to $\mathbf{p}$ is

$$KL(\mathbf{p}||\mathbf{q}) = \sum_{i=1}^{K} p_i \log \frac{p_i}{q_i} \tag{9}$$

Note that KL loss (9) is not symmetric, i.e. $KL(\mathbf{p}||\mathbf{q}) \neq KL(\mathbf{q}||\mathbf{p})$, and therefore it is not a distance metric.

C Pseudocode for the Reservoir Sampling algorithm

In this section, we provide the pseudocode for the Reservoir Sampling algorithm [18]. The key concepts of Reservoir Sampling are (i) we store a uniform subset of data points that are representative of all training data that are seen up to that time, and (ii) each incoming data point from the stream has equal probability to be stored in the buffer.

Algorithm 3 RS - Reservoir Sampling

Input: $\mathcal{M}$ is the set of buffer contents (data points); (x_*, y_*) is the incoming data point from batch $\mathcal{B}_t$ that we examine; k_t is the counter for all seen data points seen in the training data stream up to time t.

Buffer_Update($\mathcal{M}$,k_t,x_*, y_*):

- **if** $|\mathcal{M}| < M$ **then** ▷ i.e. buffer is not full
  - $\mathcal{M} \leftarrow \mathcal{M} \cup (x_*, y_*)$ ▷ Store (x_*, y_*) in the buffer
- **else**
  - Generate a random number $j \in [1, k_t]$
  - **if** $j \leq |\mathcal{M}|$ **then**
    - $\mathcal{M}[j] \leftarrow (x_*, y_*)$ ▷ Replace data point at position j with incoming data point
  - **else**
    - Ignore the data point (x_*, y_*)
  - **end if**
- **end if**