Online Continual Learning from Imbalanced Data
Supplementary Material

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

A. Proofs of Statements

Lemma A.1 Let us assume a stream that contains instances of k distinct classes, and a memory $\mathcal{M}$ of size m. If a class c consists of less than m/k stream instances, or in other words $n_c < m/k$, then it cannot be the *largest* class while the memory is *filled*.

Proof. We use proof by contradiction. Let us assume a scenario where c is a *largest* class while the memory is *filled*.[1] If we define m_{c_i} to be the number of instances of class c_i currently stored in memory, and since c is the *largest* class in memory, it is

$$m_{c_i} \leq m_c, \quad \forall i \text{ s.t. } c_i \text{ in } \mathcal{M}. \tag{1}$$

Moreover, since the memory is *filled*, it must be that

$$\sum_i m_{c_i} = m, \quad \forall i \text{ s.t. } c_i \text{ in } \mathcal{M}. \tag{2}$$

If k_m is the number of different classes present in memory, the combination of (1) and (2) gives us

$$k_m m_c \geq m. \tag{3}$$

However, since $k_m \leq k$ and $m_c \leq n_c$, from (3) we get

$$k n_c \geq m, \tag{4}$$

which cannot be true since we started from the fact that $n_c < m/k$. Thus, we proved that a class with less than m/k instances in the stream, cannot be the *largest* class when the memory is *filled*. □

Statement A.1 Let us assume a stream that contains instances of k distinct classes, and a memory $\mathcal{M}$ of size m. If a class c of the stream consists of less than m/k instances, or in other words $n_c < m/k$, then all n_c instances of said class will be stored in memory after the stream has been read.

Proof. It suffices to show that all of the n_c instances will be stored when observed, and that none of them will be overwritten.

[1] See our definitions in Subsection 2.2 of the main paper.

Regarding the first part, there are three cases applicable when an incoming instance is considered to be stored in memory (i.e., lines 4, 9, and 16 of the CBRS algorithm). In the first two (i.e., lines 4 and 9), the observed instance is always stored, and in the third case (i.e., line 16) the instance is stored with a certain probability. The third case is reached only when the memory is *filled* and when c is a *full* class. However, as we showed in the Lemma A.1, a class that consists of less than m/k instances cannot become the *largest* class when the memory is *filled*, thus it cannot be a *full* class. Therefore, the third case will never be applicable for such classes. Consequently, all encountered instances of c will correspond to one of the first two cases, and they are always stored.

Regarding the second part, there are two cases where instances stored in memory get overwritten (i.e., lines 9 and 16 of the CBRS algorithm). None of the two cases apply to stored instances of c, however, since c cannot be either *full* or the *largest* class.

Hence, we proved that all incoming instances of class c will be stored when observed, and that none of them will be overwritten by instances of another class. □

Lemma A.2 The set of currently stored instances of a class c is iid with respect to the observed stream instances of the same class, at any time step.

Proof. We use proof by induction. Let us assume a stream, of which n instances have been observed, and a memory $\mathcal{M}$ of size m. Additionally, out of all the distinct classes encountered in the stream so far, let us consider a random one, denoted by c. We denote by n_c the number of instances of class c observed thus far, and by m_c the number of instances of class c currently stored in memory. The induction takes place with respect to n.

Base case. The base case is applicable when $n = 1$, or in other words, immediately after the first stream instance is observed. The first observed instance will always be stored in memory. If the instance is of class c, it is the only one observed so far and it is present in memory. Conversely, if it is not of class c, we have neither observed nor stored any

instances of class c. Hence, whether the instance is of class c or not, the base case holds.

Inductive step. Having observed n instances of an arbitrary number of classes, we assume that all observed instances of class c have the same probability of being present in memory, and that probability is

$$p_n = \frac{m_c}{n_c}. \tag{5}$$

After the $(n+1)$-th instance is observed, we can differentiate four distinct cases for what happens next. The differentiation arises from whether the incoming instance is of class c or not, and the memory status (i.e., *largest*, *full*, or none of the above) of c.

First, we examine the case where c is not *full* and the incoming instance is of class c. According to Algorithm 1 of the main paper, the incoming instance will be stored, overwriting, in the process, an instance of another class. Since c is not *full*, we know that all of its already observed instances will be present in memory. Thus, the observed and the stored instances of c will be identical, and by extension, iid.

Second, we consider the case where c is *full* and the incoming instance is of c. We assume n_c and m_c to be the quantities we defined, before the instance is observed. Since c is *full*, the observed instance is not allowed to overwrite an instance of another class. Therefore, we need to show that all $n_c + 1$ observed instances of class c have the same probability p_{n+1} to be present in memory, where

$$p_{n+1} = \frac{m_c}{n_c + 1}. \tag{6}$$

According to Algorithm 1 of the main paper, the probability that the incoming instance will be stored is equal to p_{n+1}.[2] For an arbitrary observed instance of class c , the probability of being present in memory after the incoming instance is processed, is the product of p_n times the probability of not being overwritten by the observed instance. Specifically, it is

$$p_n \left(1 - \frac{p_{n+1}}{m_c}\right) = \frac{p_n n_c}{n_c + 1} = \frac{m_c}{n_c + 1} = p_{n+1}. \tag{7}$$

Third, in the case where c is not *full* and the incoming instance is not of c, we know that c is not the *largest* class, and thus, none of its stored instances will be overwritten. In this case, since the stored c instances were iid with respect to the observed ones, and we did not observe a c instance, the stored instances of c remain iid with respect to the observed ones.

Finally, when c is the *largest* class and the incoming instance is not of c, according to Algorithm 1 of the main paper, a stored instance of c will be overwritten. Since that c instance is selected uniformly, the probability that an arbitrary observed instance of c is present in memory after the incoming instance is processed is

$$\frac{m_c}{n_c}\left(1 - \frac{1}{m_c}\right) = \frac{m_c - 1}{n_c}. \tag{8}$$

Therefore, the stored instances of c remain iid with respect to the observed ones, because all of the observed ones have the same probability of being present in memory.

We showed that in all cases, the stored instances of an arbitrary class c remain iid with respect to the observed instances of c, after the inductive step is performed. □

Statement A.2 The stored instances of each class c are iid with respect to the stream instances of the same class, after the stream is read in whole.

Proof. Follows naturally from Lemma A.2. □

B. Extending CBRS

In the main paper, we state that in absence of any prior knowledge about the stream contents, CBRS assumes that all classes are equally important and equally difficult to learn. However, when in possession of such prior knowledge before reading the stream, CBRS can be trivially extended to take it into account. We assume a vector

$$\mathbf{i} = (i_1, i_2, \cdots, i_k), \tag{9}$$

where each i_c represents an *importance* score for class c. Moreover, we define the *extended size* of each class in memory to be m_c/i_c, where m_c is the number of elements of class c currently present in memory. CBRS, instead of using the raw size m_c, uses the *extended size* to find the *largest* class, and thus, promotes the inclusion of more samples from classes of higher importance.

In Figure 1, we present an illustration of this concept, with a memory of size $m = 500$. In the first two streams we assume that

$$\mathbf{i}_{\text{inc}} = (1, 2, \cdots, 10), \tag{10}$$

and in the remaining two we assume that

$$\mathbf{i}_{\text{dec}} = (10, 9, \cdots, 1). \tag{11}$$

We observe that CBRS is indeed taking into account the provided importance scores and boosts the storage of classes of higher importance.

[2] Note that the meanings of m_c, n_c here, are minutely different from the ones in Algorithm 1, for the sake of being more formal. Here, we assume that the values of m_c, n_c correspond to the respective quantities not taking into account the incoming instance, while in Algorithm 1, they correspond to the same quantities having taken into account the incoming instance.

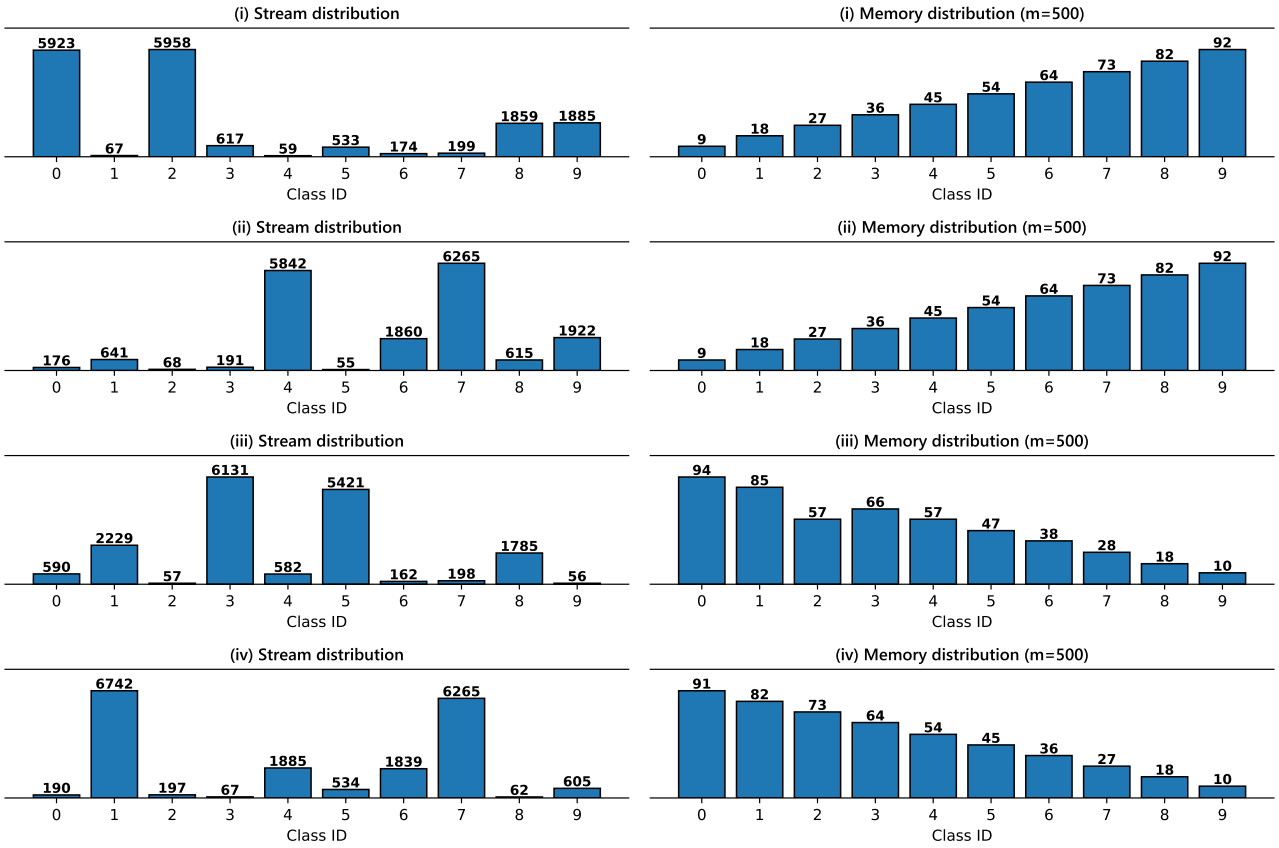

Figure 1. Extending CBRS for classes of different importance.

C. Equal Weighting of the Loss Components

We examine whether it is beneficial to use adaptive weighting of the two loss components described in Subsection 2.4 of the main paper, in comparison to always weighting both components as equal. We present the experiments of Subsections 3.4 and 3.5 with and without adaptive weighting of the loss components. All experiments are repeated ten times. We report the 95% confidence interval of the accuracy on the test set of each dataset, after the learner has been trained on the corresponding stream. The relevant results are presented in Table 1 and Table 2. We observe that using an equal loss weighting ($a = 0.5$) generally results in lower accuracy and higher variance over all different datasets and methods.

D. Different Class Orderings

At this point we examine the degree to which the class order affects the results presented in the main paper. We remind the reader, that in the main paper the learner is presented with the classes in increasing order. Here, we repeat the experiments from Subsections 3.4 and 3.5 with the class order being selected at random. All experiments are repeated ten times. We report the 95% confidence interval of the accuracy on the test set of each dataset, after the learner has been trained on the corresponding stream. The relevant results are presented in Table 3 and Table 4. The results tell roughly the same story in both cases, without having any consistent differences.

E. CBRS Illustration for Various Streams

In Figure 2, we present the memory distribution that CBRS results in, after reading four different imbalanced MNIST streams, and for three different memory sizes ($m = 100, 500, 2500$). Specifically, the first four bar plots of the figure (i.e., all plots having "(i)" in the title) depict the distribution of the first stream and the resulting memory composition for each different memory size respectively, and so on for the remaining quartets of plots.

Table 1. Comparison of memory population methods when a is adaptively selected as described in the main paper (left), and when $a = 0.5$ (right). We report the 95% confidence interval of the test set accuracy of each experiment over ten runs.

	Adaptive Loss Weighting				Equal Loss Weighting			
Methods	MNIST	F-MNIST	CIFAR-10	CIFAR-100	MNIST	F-MNIST	CIFAR-10	CIFAR-100
Naive	10.1 ± 0.0	10.0 ± 0.0	10.0 ± 0.0	1.0 ± 0.0	10.1 ± 0.1	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	30.9 ± 5.1	42.2 ± 5.6	43.0 ± 3.2	16.7 ± 1.3
Reservoir	67.9 ± 3.0	64.1 ± 1.8	54.7 ± 2.2	28.1 ± 1.2	66.3 ± 1.5	65.3 ± 2.5	49.9 ± 3.1	18.1 ± 1.0
GSS	74.2 ± 3.4	64.9 ± 3.1	63.3 ± 2.3	23.0 ± 1.1	76.8 ± 1.8	68.6 ± 2.6	52.2 ± 3.5	14.3 ± 1.9
CBRS	**83.3** ± 2.5	**75.0** ± 2.5	**73.4** ± 2.2	**40.2** ± 1.0	**81.1** ± 3.6	**73.8** ± 2.8	**63.4** ± 4.0	**22.5** ± 2.0

Table 2. Varying the memory size on Fashion-MNIST when a is adaptively selected as described in the main paper (left), and when $a = 0.5$ (right). We report the 95% confidence interval of the test set accuracy of each experiment over ten runs.

	Adaptive Loss Weighting				Equal Loss Weighting			
Methods	$m = 100$	$m = 500$	$m = 1000$	$m = 5000$	$m = 100$	$m = 500$	$m = 1000$	$m = 5000$
Random	21.7 ± 4.2	38.7 ± 5.6	57.6 ± 5.1	76.1 ± 1.3	20.5 ± 4.5	37.1 ± 4.7	53.5 ± 3.3	74.5 ± 3.1
Reservoir	53.7 ± 3.0	64.3 ± 2.8	68.2 ± 2.0	75.2 ± 1.9	52.5 ± 2.9	63.6 ± 3.0	66.0 ± 2.6	73.8 ± 4.3
GSS	59.5 ± 3.6	65.0 ± 3.8	65.1 ± 3.9	70.7 ± 3.2	59.1 ± 1.6	65.6 ± 2.9	70.2 ± 2.1	72.3 ± 3.0
CBRS	**66.5** ± 1.5	**75.8** ± 2.0	**76.7** ± 2.4	**77.1** ± 1.8	**65.9** ± 1.5	**74.9** ± 2.5	**75.1** ± 2.7	**76.5** ± 3.7

Table 3. Comparison of memory population methods when the classes are presented in increasing order (left), and when they are presented in random order (right). We report the 95% confidence interval of the test set accuracy of each experiment over ten runs.

	Increasing Class Order				Random Class Order			
Methods	MNIST	F-MNIST	CIFAR-10	CIFAR-100	MNIST	F-MNIST	CIFAR-10	CIFAR-100
Naive	10.1 ± 0.0	10.0 ± 0.0	10.0 ± 0.0	1.0 ± 0.0	10.3 ± 0.5	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	33.0 ± 5.6	37.3 ± 7.7	50.1 ± 4.9	25.2 ± 0.9
Reservoir	67.9 ± 3.0	64.1 ± 1.8	54.7 ± 2.2	28.1 ± 1.2	70.4 ± 2.5	64.6 ± 2.1	56.1 ± 2.4	28.8 ± 0.8
GSS	74.2 ± 3.4	64.9 ± 3.1	63.3 ± 2.3	23.0 ± 1.1	75.8 ± 3.8	66.5 ± 2.3	64.3 ± 1.8	24.0 ± 0.9
CBRS	**83.3** ± 2.5	**75.0** ± 2.5	**73.4** ± 2.2	**40.2** ± 1.0	**85.8** ± 1.2	**76.1** ± 1.8	**74.4** ± 2.4	**39.7** ± 0.9

Table 4. Varying the memory size on Fashion-MNIST when the classes are presented in increasing order (left), and when they are presented in random order (right). We report the 95% confidence interval of the test set accuracy of each experiment over ten runs.

	Increasing Class Order				Random Class Order			
Methods	$m = 100$	$m = 500$	$m = 1000$	$m = 5000$	$m = 100$	$m = 500$	$m = 1000$	$m = 5000$
Random	21.7 ± 4.2	38.7 ± 5.6	57.6 ± 5.1	76.1 ± 1.3	28.5 ± 5.1	43.1 ± 6.1	57.6 ± 5.2	72.3 ± 2.6
Reservoir	53.7 ± 3.0	64.3 ± 2.8	68.2 ± 2.0	75.2 ± 1.9	54.6 ± 2.1	65.9 ± 2.9	66.6 ± 1.7	72.8 ± 2.7
GSS	59.5 ± 3.6	65.0 ± 3.8	65.1 ± 3.9	70.7 ± 3.2	62.5 ± 2.7	66.8 ± 2.1	66.2 ± 2.8	68.8 ± 3.4
CBRS	**66.5** ± 1.5	**75.8** ± 2.0	**76.7** ± 2.4	**77.1** ± 1.8	**66.5** ± 2.8	**73.3** ± 1.7	**74.6** ± 2.4	**75.7** ± 2.0

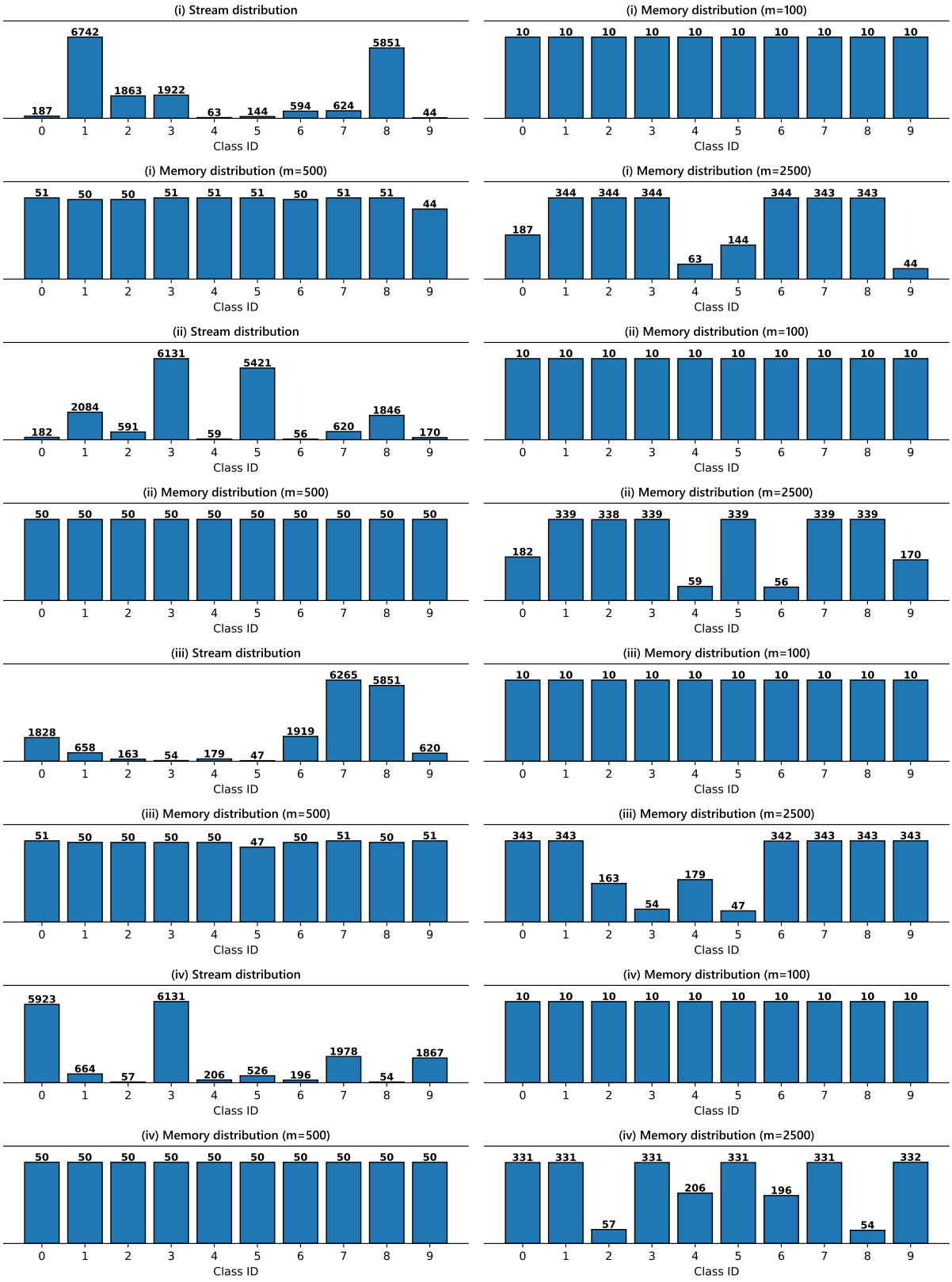

Figure 2. Resulting CBRS memory distributions (for three different memory sizes $m = 100, 500, 2500$) after being presented with four different imbalanced streams.