Generative negative replay for continual learning

Generative Negative Replay for Continual Learning

Gabriele Graffieti[a,*], Davide Maltoni[a], Lorenzo Pellegrini[a], Vincenzo Lomonaco[b]

[a] *Department of Computer Science and Engineering, University of Bologna, Italy*
[b] *Department of Computer Science, University of Pisa, Italy*

Abstract

Learning continually is a key aspect of intelligence and a necessary ability to solve many real-life problems. One of the most effective strategies to control catastrophic forgetting, the Achilles' heel of continual learning, is storing part of the old data and replaying them interleaved with new experiences (also known as the replay approach). Generative replay, which is using generative models to provide replay patterns on demand, is particularly intriguing, however, it was shown to be effective mainly under simplified assumptions, such as simple scenarios and low-dimensional data. In this paper, we show that, while the generated data are usually not able to improve the classification accuracy for the old classes, they can be effective as negative examples (or antagonists) to better learn the new classes, especially when the learning experiences are small and contain examples of just one or few classes. The proposed approach is validated on complex class-incremental and data-incremental continual learning scenarios (CORe50 and ImageNet-1000) composed of high-dimensional data and a large number of training experiences: a setup where existing generative replay approaches usually fail.

Keywords: Continual Learning, Generative Replay, Continual Object Recognition, Pseudo-Rehearsal, Generative Model, Negative Replay

*Corresponding author
Email addresses: gabriele.graffieti@unibo.it (Gabriele Graffieti), davide.maltoni@unibo.it (Davide Maltoni), l.pellegrini@unibo.it (Lorenzo Pellegrini), vincenzo.lomonaco@unipi.it (Vincenzo Lomonaco)

Published in Neural Networks:
https://doi.org/10.1016/j.neunet.2023.03.006.

1. Introduction

The majority of neural network training approaches assume that is feasible to build a set of independent and identically distributed (i.i.d.) samples to train the model. This assumption is in contrast with biological learning since intelligent beings observe the world as an ordered sequence of highly correlated data. When state-of-the-art deep neural networks are trained continually, and the whole data cannot be accessed at once, the model suffers from the catastrophic forgetting problem (McCloskey & Cohen, 1989), and the knowledge about old data (old experiences) tend to be overwritten by new examples.

Several continual learning (CL) approaches have been recently proposed to improve continual learning in artificial neural networks (see Delange et al. (2021); Maltoni & Lomonaco (2019); Parisi et al. (2019) for comprehensive surveys). Replay methods (see the in-depth review by Hayes et al. (2021)) usually perform better than other approaches, and in some complex continual learning scenarios replay seems to be the only strategy capable of mitigating catastrophic forgetting (van de Ven et al., 2020). However, due to the need of storing old examples, replay methods are not appropriate if past data cannot be memorized for privacy or security reasons. Moreover, the memory and computation overhead can pose issues, especially in edge devices (Pellegrini et al., 2021), or when the number of experiences is very large.

Therefore, *generative replay* has been explored, where a generative model is trained to produce data from past experiences (see Lesort et al. (2018) and Shin et al. (2017)). Besides solving the replay memory issue, generative replay can theoretically be capable of generating more general and novel examples not included in past experiences, thus potentially overcoming replay methods. Unfortunately, generative replay introduces much complexity due to the need for an interleaved incremental training of both a classifier and a generator. Moreover, generative models are usually complex and unstable to train, especially in incremental scenarios. Several researchers have shown that generative replay fails in complex CL scenarios with high-dimensional data (see Aljundi et al.

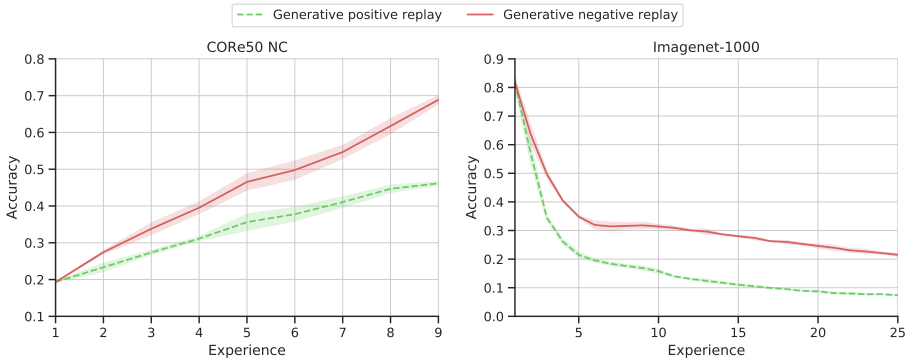

Figure 1: The proposed generative negative replay is compared with classical generative replay on two complex class incremental CL benchmarks (details in section 4.2). In both the benchmarks, using the same classifier, generator, and training procedure, negative replay performs significantly better.

(2019); Lesort et al. (2018) and van de Ven et al. (2020)) mainly due to the inaccuracies in the data generation that progressively grows across the experiences if a single generator is incrementally updated. The *photocopy example* helps to understand why. Let us consider a high-quality photocopy machine: when a picture is initially copied the output looks very similar to the original, but if the process is repeated several times by using as input the output of the previous step, some artifacts will soon appear and, after many iterations, the result will be highly compromised. Hence, even if some state-of-the-art models have been proved to be effective in generating also high dimensional data (Huang et al. (2018) Karras et al. (2019)), the continual training of such generators remains a challenging problem.

Although generative models are hot research topics and we can expect improved methods in the future, as of today we must deal with imperfect generated data and try to exploit them at best when a classifier is incrementally trained. The proposed approach, denoted as *Generative Negative Replay*, does not attempt to improve the knowledge of old classes using the generated data because it assumes that the data quality is not high enough for this purpose. Nevertheless, it makes use of generated (latent) data as negative examples to

better learn the classes of current experience, especially when the number of classes per experience is small and we incur in the "learning in isolation" problem.

We experimentally demonstrate, on complex benchmarks such as CORe50 and ImageNet-1000, where (positive) generative replay fails, that negative replay is effective to contrast the learning in isolation problem, allowing to train a classifier incrementally across a high number of experiences (see Figure 1). We also investigate the impact of data quality on negative replay by running experiments (section 4.5) where negative examples are sampled from original past patterns (upper bound) and randomly generated.

2. Problem Formulation

A continual learning (CL) problem consists of a number N_E of experiences, each containing a subset of data that is only accessible during the corresponding experience:

$$\text{CL} = \{e_1, e_2, ..., e_{N_E}\}. \tag{1}$$

Each experience is composed of several data points and the corresponding labels:

$$e_k = (\mathcal{X}_k, \mathcal{Y}_k), \;\; \mathcal{X}_k = \{x_1^k, x_2^k, ..., x_{N_k}^k\}, \; \mathcal{Y}_k = \{y_1^k, y_2^k, ..., y_{N_k}^k\} \tag{2}$$

where x_i^k and y_i^k are the data points and the associated labels contained in the k-th experience and N_k is the number of samples in the k-th experience.

Let $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ be the entire dataset, then $\mathcal{X} = \bigcup_{i=1}^{N_E} \mathcal{X}_i$ and $\mathcal{Y} = \bigcup_{i=1}^{N_E} \mathcal{Y}_i$. We can define three different scenarios for supervised continual learning (Maltoni & Lomonaco, 2019; van de Ven & Tolias, 2018) based on the labels $\mathcal{Y}_k$ contained in the experiences ($k \in \{1, ..., N_E\}$ with N_E the total number of experiences) as follows:

New instances (NI) also known as domain-incremental learning (Domain-IL), where all the labels are known from the first experience, and in the successive experiences, only new instances of the same classes are included. Formally, we could define the NI scenario as:

$$\mathcal{Y}_1 \cap \mathcal{Y}_k = \mathcal{Y}_k \quad \text{for } k = \{1, ..., N_E\}, \tag{3}$$

meaning that every possible label of the entire dataset must be present in the first experience.

New classes (NC) also known as class-incremental learning (Class-IL), where each experience includes data of classes not present in any other past experience. Formally, we can define the NC scenario as:

$$\mathcal{Y}_k \cap \bigcup_{i=1}^{k-1} \mathcal{Y}_i = \emptyset \quad \text{for } k = \{2, ..., N_E\}. \tag{4}$$

New instances and classes (NIC) where a new experience can contain already seen classes, new classes, or a mix of the two. This is the most natural scenario since in the real world an agent may sense both known and unknown objects. Formally the NIC scenario can be defined as:

$$\exists k : \mathcal{Y}_k \cap \bigcup_{i=1}^{k-1} \mathcal{Y}_i \neq \emptyset \text{ and } \exists j : \mathcal{Y}_1 \cap \mathcal{Y}_j \neq \mathcal{Y}_j. \tag{5}$$

Meaning that there is at least one experience that contains classes already seen in the past (left part) and at least one experience that contains classes not present in the first experience (right part).

Given the above definitions, our goal is to fit a function f, parametrized by Θ, to the sequence of experiences. A naive approach is finding the best parameters Θ^* that minimizes:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(f_\Theta(\mathcal{X}_i), \mathcal{Y}_i) \text{ for } i = \{1, ..., N_E\}, \tag{6}$$

where $\mathcal{L}(\cdot)$ is a loss function (e.g. cross-entropy loss).

As first pointed out by McCloskey & Cohen (1989), this simple approach is prone to catastrophic forgetting, thus the model f_Θ is not able to learn the experiences $\{e_1, e_2, ..., e_{N_E}\}$ sequentially.

2.1. Continual learning with replay

Replay is an effective approach to overcome the catastrophic forgetting problem (van de Ven et al., 2020; Hayes et al., 2021). It consists in storing into

a memory $\mathcal{M}$ a subset of past data and using them jointly with the data of the current experience for the model optimization. In presence of replay, Equation 6 becomes:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(f_\Theta(\mathcal{X}_i \cup \mathcal{M}_i^x), \mathcal{Y}_i \cup \mathcal{M}_i^y) \text{ for } i = \{1, ..., N_E\}, \tag{7}$$

where $\mathcal{M}_i^x$ and $\mathcal{M}_i^y$ are the datapoints and labels contained in the replay memory during the training on experience i. During the first experience we have that $\mathcal{M}_1^x = \mathcal{M}_1^y = \emptyset$.

From Equation 7 is evident that replay has two main issues: space and time. Space since storing old data require memory (for high dimensional data and a large number of experiences the memory required may be intractable), and time since in every experience the model needs to be updated also with the data contained into $\mathcal{M}$, leading to extra computations.

2.2. Continual learning with generative replay

To overcome the aforementioned issues generative replay can be used. Generative replay requires to train simultaneously and incrementally a classifier and a generative model (Shin et al., 2017; Wu et al., 2018; Thandiackal et al., 2021). The generative model g, parametrized by Ω provides surrogate data similar to the past experiences' data. In the case of a conditional generative model (in which we can control the class of the generated data), the optimal parameters of the classifier can be derived using Equation 7, with the difference that the replay memory $\mathcal{M}$ is populated as:

$$\mathcal{M}_i^x \leftarrow g_\Omega(z_j|y_j); \;\; \mathcal{M}_i^y \leftarrow y_j; \;\; y_j \in \bigcup_{k=1}^{i-1} \mathcal{Y}_k, \;\; j = \{1, ..., R\}, \tag{8}$$

where R is the number of generated replay patterns (size of memory), z_j is a latent random input vector given to the generative model, y_j is a label sampled from the set of labels encountered in the past experiences, and "$\leftarrow$" indicates the insertion of an element into the memory.

Some solutions to continually train a generative model have been proposed in the literature, as discussed in section 5, but the problem is still far to be

solved (Lesort et al., 2018; van de Ven et al., 2020; Mundt et al., 2019) when the number N_E of experiences is large and the data is high dimensional.

One solution is that the same generated data fed to the classifier can be used to control forgetting in the generative model as well. Instead of a generic generative model, let us suppose we have a conditional generative model composed of an encoder q_γ parametrized by γ and a decoder p_ξ parametrized by ξ, such that $g_\Omega = p_\xi \circ q_\gamma,\ \Omega = (\gamma, \xi)$. In case of replay, we can maintain the generative model's parameters of the previous experience $\Omega' = (\gamma', \xi')$ and use them to generate replay patterns by sampling a latent random vector z, conditioning it to a previous class, and then populating the replay memory as:

$$\mathcal{M}_i^x \leftarrow p_{\xi'}(z_j|y_j);\ \ \mathcal{M}_i^y \leftarrow y_j;\ \ y_j \in \bigcup_{k=1}^{i-1} \mathcal{Y}_k,\ \ j = \{1, ..., R\}, \tag{9}$$

with z_j sampled from the encoder target distribution. The optimal parameters of the generative model can thus be obtained requiring that the generated data are similar (L2 loss) to the original ones:

$$\gamma^*, \xi^* = \arg\min_{\gamma,\xi} \|p_\xi(q_\gamma(\mathcal{X}_i \cup \mathcal{M}_i^x)|\mathcal{Y}_i \cup \mathcal{M}_i^y) - \mathcal{X}_i \cup \mathcal{M}_i^x\|_2^2 \quad \text{for } i = \{1, ..., N_E\}, \tag{10}$$

where $q_\gamma(\mathcal{X}_i)$ is forced to follow a target distribution, typically $\mathcal{N}(0, 1)$.

3. Generative negative replay

Generative replay is an appealing strategy for continual learning, but, to exploit it in complex scenarios with many experiences, we need to overcome the data degradation issue. Since this problem is not easily addressable on the generator side, we propose to circumvent it by changing the way the classifier makes use of generated data.

Let us suppose the classifier f_Θ can be divided into a feature extractor f_ϕ, parametrized by ϕ and a classification head c_ψ parametrized by ψ, so that $f_\Theta = c_\psi \circ f_\phi,\ \Theta = (\phi, \psi)$. In our case, the feature extractor ϕ is a convolutional neural network, while the classification head ψ is a single fully connected layer.

In any case, every possible gradient-based model can be used in conjunction with our proposal. The parameters ψ of the classification head can be divided into C groups, where C is the number of classes. The groups, denoted as (ψ^1, ψ^2, ..., ψ^C) represents the parameters associated to the connections between the features extracted by f_ϕ and the output neuron of the corresponding class.

For simplicity, let us assume that the feature extraction weights ϕ are frozen (after an initial pre-training) and, across the experiences, we only learn the classification head weights ψ. As explained in section 3.3, this assumption is not necessary and our experiments were carried out by learning both ϕ and ψ.

3.1. Learning classes in isolation

Learning in isolation is one of the main causes of catastrophic forgetting, especially in the NC or NIC scenarios where only a limited number of classes are present in a single experience, and the parameters of the classification head are learned without negative examples that counteract the "greediness" of the optimization. As an example, let us consider an NC scenario where only one class is present in each experience. Suppose that c is the only class in the experience k, then the best way to optimize the model is to change the parameters ψ^c to maximize the output of the output neuron c for every input and change the rest of $\psi^j,\ j \neq c$ to minimize the output for the remaining classes. This still holds if in the experience are present only a few classes, since the model is only optimized to discriminate between the present classes and has no interest in maintaining the past acquired knowledge.

3.2. Positive and negative replay

Replay can be used to counteract the learning in isolation problem, however, when the replay data comes from a generative model, the data quality degradation has a negative impact on the classifier training. The aforementioned problem is typical of the standard generative replay approach (hereafter denoted as *generative positive replay*), where replay data is used by the classifier in the same

manner of the current experience's data, and therefore the classification head's weights associated to the replay classes are optimized based on the replay data.

On the contrary, in the proposed *generative negative replay* approach, the replay patterns are used to counteract the detrimental effects of the training in isolation, but they are not used to modify the parameters ψ associated with the replay classes. The key idea (validated experimentally) is that the generated patterns are valid antagonists to mitigate the learning in isolation problem, but their quality is not enough to improve the knowledge of classes originally learned on real data. It is well known that one class learning approaches are in general less effective than discriminative learning because the presence of both positive and negative examples allows to better characterize the classification boundaries (Hempstalk & Frank, 2008). Therefore, the proposed approach exploits generated data to constrain the classification boundary and to avoid the real data in the current experience pulling it too much in their direction.

3.3. Training a classifier with generative negative replay

The idea of generative negative replay is quite general and can be used in conjunction with different continual learning classification approaches and scenarios (NI, NC, NIC). To avoid replay data (i.e. negative examples) altering the knowledge of the already learned classes, the gradient propagation can be selectively blocked during the backward pass. The general idea is illustrated in Figure 2. While the original examples ($\mathcal{X}_i$) normally flow forward and backward[1] throughout the model, the replay examples ($\mathcal{M}_i^x$) are passed forward, but, before the backward pass, the loss tensor is masked at the class level by resetting the gradient components corresponding to the classes in $\mathcal{M}_i^y$. In this case, the gradient of the possibly degraded replay data does not contribute to the learning of class boundaries performed in the classification head ψ. This is done since

[1] In this context, for *forward* we mean the flow of the data through the model, while for *backward* we mean the flow of gradients after loss calculation in the reverse direction of the data.

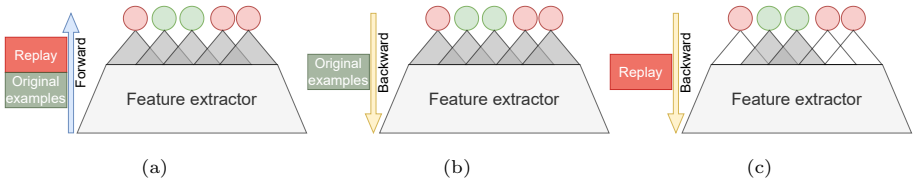

Figure 2: Graphical representation of the negative replay idea. Green output neurons represent the classes present in the current experience, while red output neurons represent the replay classes. During forward (a) both the replay data and the original data from the current experience flow through the network. During backward, the original data gradient flow through all the neurons of the classification head (b), while the replay data contribution is masked and the gradient only flows through the neurons of classes belonging to the current experience (c).

the quality of replay data may be not high enough to correctly improve the learning of old class boundaries. Nonetheless, the gradient of the replay data flows through the output neurons associated with current data to contrast the learning in isolation problem. This gradient also flows through the feature extractor ϕ and helps to also improve it. Such a masking-based negative replay has been implemented and tested in conjunction with two well-known continual learning approaches, such as *experience replay* (ER) (Chaudhry et al., 2019), and *Learning without Forgetting* (LwF) (Li & Hoiem, 2016) (see section 4.4).

Hereafter, we propose an alternative implementation embedded in the AR1 algorithm (Maltoni & Lomonaco, 2019), whose update mechanism for the classification head weights allows very simple and efficient integration of negative replay. AR1 is a flexible continual learning approach that can achieve state-of-the-art accuracy on complex CL benchmarks. In Appendix B, AR1 is shown to outperform several well-known CL algorithms on the complex ImageNet-1000 benchmark proposed by Masana et al. (2022).

AR1 uses different mechanisms to learn the classification head and the feature extractor weights. The feature extraction weights ϕ are protected against forgetting: i) through the Synaptic Intelligence (SI) regularization technique (Zenke et al., 2017) or ii) using a replay memory with a small learning rate (denoted as AR1free in Pellegrini et al. (2020)). The classification head weights

ψ are managed by CWR (Maltoni & Lomonaco, 2019). CWR is a simple method aimed at addressing the score bias problem produced by imbalance learning during continual learning (Belouadah et al., 2020). The score bias problem is caused by the imbalance between classes in the current data and classes present only in the replay data. Often, classes in the present experience are represented by more numerous examples than classes only present in the replay memory, producing a bias towards the former. This frequently leads to the forgetting of old knowledge in favor of current data. To address this problem, CWR maintains a copy of the weights of the classification head of the previous experience (ψ') and at the start of each experience the classification head is reset and only weights of classes of the current experience are loaded from ψ'. At the end of the experience, a weight consolidation phase takes place, where the weights ψ learned in the current experience are consolidated with the weights ψ'. This procedure helps to contrast the aforementioned score bias problem, since the weights associated with classes in the current experience are "averaged" using weights from past experiences, mitigating every bias than can appear during the training.

During the weight consolidation phase the negative and positive replay differ. In particular, for each parameter group ψ^c associated to a class c belonging to the current experience plus the current memory ($c \in \mathcal{Y}_k \cup \mathcal{M}_k^y$), the mean of all the parameter group $\mu(\psi^c)$ is calculated, and subtracted to all the parameters in the group, to force zero mean: $\psi^c = \psi^c - \mu(\psi^c)$. This prevents class bias problems due to the different magnitudes of the weights. Then, there are three possibilities, based on c:

1. c is a new class never seen before ($c \in \mathcal{Y}_k \wedge c \notin \bigcup_{i=1}^{k-1} \mathcal{Y}_i$): in this case ψ^c is maintained as is.
2. c is a class seen before ($c \in \mathcal{Y}_k \wedge c \in \bigcup_{i=1}^{k-1} \mathcal{Y}_i$): the consolidation step is applied, so $\psi^c = \frac{\psi'^c \cdot w_{past_c} + \psi^c}{w_{past_c}+1}$ where w_{past_c} is a parameter that balances the contribution of the past w.r.t. the present, calculated as follows:

$$w_{past_c} = \sqrt{\frac{past_c}{current_c}}, \tag{11}$$

where $past_c$ is the number of data points of class c encountered in past experiences, while $current_c$ is the number of data points of class c present in the current experience.

3. c is not in the current experience but is a replay example ($c \notin \mathcal{Y}_k \wedge c \in \mathcal{M}_k^y$):
   - in case of positive replay apply consolidation (step 2).
   - in case of negative replay ψ^c is reverted back to ψ'^c (no contribution to the parameters ψ^c from replay examples).

The pseudo-code of the above weights consolidation algorithms is reported in Algorithm 1. It is worth noting that, in the proposed embedding of negative replay in AR1, the replay pattern can alter the feature extraction weights since CWR weight consolidation only "protects" the classification head However, in our experiments, we found that a more complex embedding of negative replay in AR1 where we block the gradient propagation for negative patterns throughout the feature extraction layers performs very similarly, and therefore we opted for simplicity.

4. Experiments and results

In this section, we describe the experimental setup used to validate the proposed negative replay. We focus on difficult continual learning scenarios, where data is high-dimensional, non-i.i.d. and the number of experiences is very large. Negative replay, implemented on top of the three CL strategies (ER, LwF, and AR1) is compared with alternative strategies (e.g. positive replay) and the role of quality of generated data is investigated by also using, as negative replay patterns, real and random data.

4.1. Experimental setup

Datasets. We performed our experiments on the CORe50 dataset (Lomonaco & Maltoni, 2017) and ImageNet-1000 dataset (Deng et al., 2009). CORe50 dataset was specifically collected for continual learning (NI, NC, and NIC scenarios) and is composed of small video sessions (about 300 frames) of 50 objects taken from

Algorithm 1 Weight consolidation

Require: ψ, ψ', $\mathcal{Y}_e$, $\mathcal{M}_e^y$

for each class $c \in \mathcal{Y}_e \cup \mathcal{M}_e^y$ **do**
  $\psi^c = \psi^c - \mu(\psi^c)$
  if $c \in \mathcal{Y}_e \wedge c \in \bigcup_{i=1}^{e-1} \mathcal{Y}_i$ **then**
    $\psi^c = \frac{\psi'^c \cdot w_{past_c} + \psi^c}{w_{past_c} + 1}$
  end if
  if $c \notin \mathcal{Y}_e \wedge c \in \mathcal{M}_e^y$ **then**
    if positive replay **then**
      $\psi^c = \frac{\psi'^c \cdot w_{past_c} + \psi^c}{w_{past_c} + 1}$
    end if
    if negative replay **then**
      $\psi^c = \psi'^c$
    end if
  end if
end for
$\psi' = \psi$

an egocentric view. Every class has 11 video sessions (a total of about 3,300 images) with different backgrounds and illuminations. Eight video sessions for each class are used for training, and three for testing. Images have size 128×128 pixels. ImageNet is composed of 1,000 classes with about 1,000 patterns per class for training and 100,000 images for testing. All images are resized to 224×224 pixels.

Classifier architecture. In the experiments with the CORe50 dataset we follow Maltoni & Lomonaco (2019) and Lomonaco et al. (2020) by employing a MobileNetV1 network (Howard et al., 2017). As suggested by Pellegrini et al. (2020) and van de Ven et al. (2020), we opted for latent replay, that is replaying latent activations instead of input data. As described in Pellegrini et al. (2020), the choice of the latent replay layer is related to a tradeoff between accuracy

and efficiency. For CORe50 experiments, as in Pellegrini et al. (2020), we used the `conv5_4` layer as latent replay layer, and the classifier was pretrained on ImageNet-1000. We also substituted all the batch normalization layers of the network with batch renormalization (Ioffe, 2017). For ImageNet-1000 we use a ResNet-18 (He et al., 2016) architecture. Following the benchmark proposed by Masana et al. (2022) the model was not pretrained. To maintain compatibility with the experiments on CORe50, even on ImageNet-1000 we use latent replay, setting the replay layer on the fourth residual block of the network (after `conv4_x` using He et al. (2016) nomenclature) The above specifications apply to all three continual learning algorithms tested. It is worth noting that:

- The experience replay approach fine-tunes the model throughout the experiences with no specific protection against forgetting, except the replay. Negative replay was implemented according to the gradient masking approach (see Figure 2). Without using replay, the ER approach becomes the *naive* approach described in (Maltoni & Lomonaco, 2019).

- LwF (Li & Hoiem, 2016) extends the loss by introducing a distillation component that regularizes the model being tuned by forcing it to produce stable outputs on past data. Here too negative replay was implemented according to the gradient masking approach (see Figure 2).

- AR1 was used with *Synaptic Intelligence* (SI) (Zenke et al., 2017) regularization when trained without replay, and without protection on the feature extraction weights (AR1free) in case of positive and negative replay (Pellegrini et al., 2020). Positive or negative replay was embedded in CWR as discussed in section 3.3.

Generative model architecture. For the choice of a generative model, we initially focused on three state-of-the-art approaches whose implementations are open source (van de Ven et al., 2020; Shin et al., 2017; Ayub & Wagner, 2021). However, since they were designed to work in simpler settings (with a lower data dimensionality and a smaller number of experiences), we were not able to port

and scale them to our complex setups. Therefore, we implemented a generative model by trying to combine the most promising techniques and ideas from different sources and control its overall memory/computation complexity. In particular, taking inspiration from van de Ven et al. (2020) we use a Variational Autoencoder (VAE) model (Kingma & Welling, 2014), but unlike van de Ven et al. (2020) we opted for a conditional VAE (cVAE) configuration (Sohn et al., 2015). Moreover, we partially blend the generator (encoder) with the classifier model: both the networks share the same feature extractor f_ϕ. Sharing part of the model between the classifier and the generator may cause some problems since the updates of the parameters performed by one model can harm the performance of the other. Nevertheless, we empirically observed that sharing only the first layers does not degrade the performance during alternate updates since the changes performed to the initial layers are minimal and not disruptive.

Finally, instead of generating raw data, we generate activations at an intermediate "latent" level as suggested by van de Ven et al. (2020). A detailed discussion on the architecture of the generator is provided in Appendix A, including a pseudo-code that highlights the details of the interleaved training of the generator and the classifier.

4.2. Experiments on the NC scenario

The first round of experiments has been performed using the AR1 algorithm on the NC scenario using CORe50 and ImageNet-1000. For CORe50 the benchmark is composed of 9 experiences: the first one contains 10 classes while the following contains five classes each. We used a class-balanced replay memory of 1,500 patterns, and (for generative replay) we inserted in each minibatch, of size 128, 14 replay patterns, and 114 patterns from the current experience. We train both the classifier and the generator for 4 epochs for each experience. Hyper-parameters of the classifier and generator are reported in Appendix C and Appendix D respectively.

For ImageNet-1000 the benchmark follows the one proposed by Masana et al. (2022): the dataset is divided into 25 experiences of 40 classes each. We used a

class-balanced replay memory of 20,000 patterns, and (for generative replay) we inserted in each minibatch, of size 128, 36 replay patterns, and 92 patterns from the current experience. We did not expect negative replay to perform well in this setup, because each experience already contains 40 classes and, therefore, the learning-in-isolation problem is here marginal. Nevertheless, we were interested in understanding if, in this setup, negative replay hurts the learning process or still provides some benefits.

The results are shown in Figure 3 and Table 1. In CORe50 the baseline with no replay (using the AR1 algorithm) reaches a final accuracy of about 60% while using replay raises the accuracy to more than 70% (Positive Replay Original Data - PR-OD). These were expected to be the lower and upper bounds of this experiment, respectively. However, because of the data degradation problem, performing positive replay with generated data (Positive Replay Generated Data - PR-GD) performed significantly worse than the case with no replay. Using replay in a negative manner with generated data, as proposed in this work (NR-GD), only slightly decreases the final accuracy w.r.t. the upper bound PR-OD.

Method	**CORe50**	**ImageNet-1000**
No Replay	41.68 ± 0.62	31.91 ± 0.17
PR-OD (upper bound)	47.02 ± 0.45	38.02 ± 0.08
PR-GD	34.05 ± 0.29	18.29 ± 0.07
NR-GD	$\mathbf{44.63 \pm 0.77}$	$\mathbf{32.74 \pm 0.17}$

Table 1: Average accuracy on all the experiences for the CORe50 and ImageNet-1000 NC scenarios.

For ImageNet-1000, due to the complexity of the experiment and the fact that the network is fully trained only during the first experience (blocked after `conv4_x` in the following experiences) the final accuracy are quite similar for all the methods (except PR-GD that performed far worse). However, in the first 10 experiences some differences can be appreciated: see the inset view in

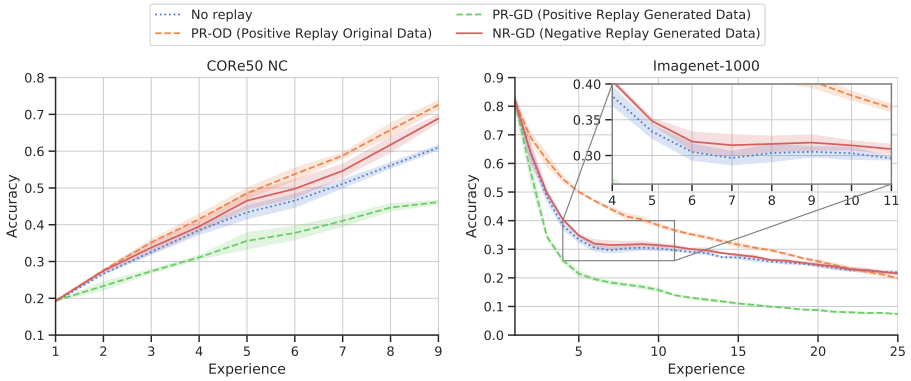

Figure 3: Overall accuracy on CORe50 NC scenario, using the whole test set (even at intermediate experiences) as defined in the CORe50 protocol (Lomonaco & Maltoni, 2017) (left), and on ImageNet-1000 using a growing test set as defined by Masana et al. (2022) (right). For a direct comparison of the two benchmarks, a plot of the experiments on CORe50 NC using a growing test set is included in Appendix Appendix E. Every experiment is averaged over 3 runs using different seeds and class order. The standard deviation is reported in light colors. Better viewed on a computer monitor.

Figure 3-right. The impact of the generated data quality on negative replay is more evident in Table 1: using negative replay with generated data (in this case highly degraded) improve the average accuracy (calculated as the mean of the accuracy after each experience) of more than 24 points and the final accuracy of more than 10 points w.r.t. using replay data in a positive manner. Furthermore, even if in this scenario the advantage of negative generative replay is little with respect to the no replay case, we note that negative replay is not hurting the training process even in scenarios where learning in isolation is only a minor issue.

4.3. Experiments on the NIC benchmark

AR1 algorithm was here tested on CORe50 NIC-391 protocol, which is composed of 391 learning experiences, each containing examples of a single class (300 frames of a short video). This scenario is particularly challenging and prone to learn-in isolation issues, hence we may expect the role of replay to be

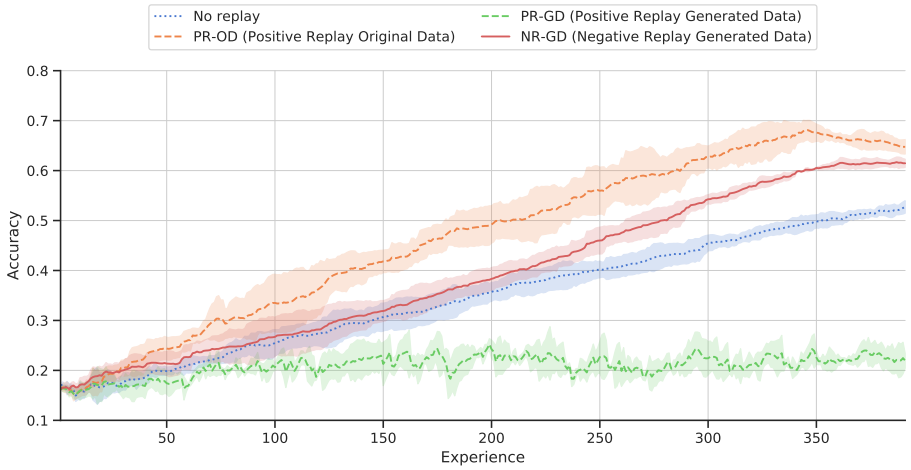

Figure 4: Overall accuracy on CORe50 NIC391 scenario, using the whole test set as defined in the CORe50 protocol (Lomonaco & Maltoni, 2017). Every experiment is averaged over 3 runs using different seeds and class order. The standard deviation is reported in light colors. Better viewed on a computer monitor.

more important here. In this scenario, we used a replay memory of only 300 patterns. The minibatch size is 128, and when generative replay is employed, we generate 64 patterns for every mini-batch (plus 64 from the current experience). Hyper-parameters of the classifier and generator are reported in Appendix C and Appendix D respectively.

The results are shown in Figure 4 and they are quite in line with the previous experiment, but here the accuracy gaps grow and the benefit of replay is more evident. The proposed negative replay with generated data (NR-GD) performs quite well, about 10 points better than no replay and just less than 5 points worse than positive replay with real data, the upper bound. In the latter case, a decline in performance after 350 experiences is visible. This can be explained by the distribution of class patterns throughout the NIC 391 experiences, which can lead to a sort of saturation in the last 30-40 experiences when all the classes have been already introduced, and only new instances of existing classes are provided. As we expect, using generated data in a positive manner (PR-GD) is

here even worse than in the NC case, because the data degradation is amplified during so many learning iterations: PR-GD is losing 30 points w.r.t. not using replay at all, and performs about 40 points worse than using the same replay data with the proposed generative negative replay approach.

4.4. Comparing negative replay across different strategies

This section aims to show that the negative replay idea is somewhat algorithm agnostic, and can bring benefits to other CL approaches (besides AR1). Therefore, we repeated the test on the CORe50 NC scenario for the ER and the LwF algorithms (Figure 5-left and 5-center, respectively), maintaining unchanged the generative model and the training dynamics. In Figure 5-right we report again AR1 results for the sake of comparison. The hyper-parameters of the strategies are reported in Appendix C. As expected the accuracy of ER is lower than LwF and, consistently with Maltoni & Lomonaco (2019), the accuracy of LwF is lower than AR1. However, all the approaches benefit from negative replay, whose accuracy is higher than no replay and Positive Replay with Generated Data (PR-GD). In particular, even the ER approach, which has no specific protection against class bias and could be hassled by the gradient masking of negative classes, shows a consistent improvement. Finally we observe that, for LwF, the gap between negative and positive is smaller than for the other algorithms, probably because distillation makes this approach quite robust w.r.t. generated data quality. On the other hand, the accuracy of LwF with both positive and negative replay is more than 10 points lower than AR1 with negative generative replay.

4.5. Negative replay with original and random data

The effect of generated data quality on negative replay is investigated by performing two further experiments using AR1: NR-OD uses original data (max. quality) for negative replay, while NR-RD uses randomly generated replay data, obtained by uniform random sampling in the latent replay layer and assigning to each data point a random class label. Since in our experiments we

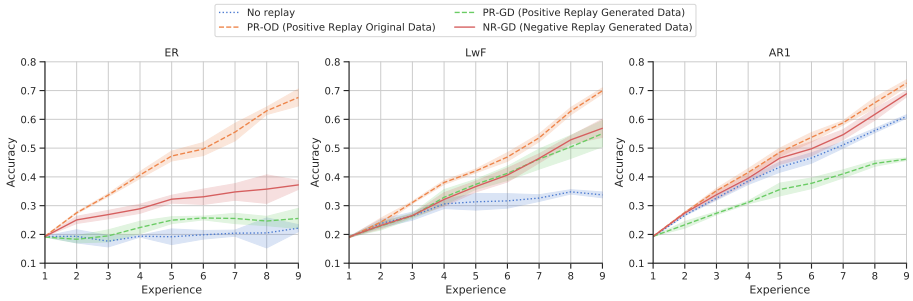

Figure 5: Overall accuracy on CORe50 NC scenario, using the whole test set as defined in the CORe50 protocol (Lomonaco & Maltoni, 2017), for ER, LwF, and AR1. Every experiment is averaged over 3 runs using different seeds and class orders. The standard deviation is reported in light colors. Better viewed on a computer monitor.

replay hidden features, to produce reasonable replay data we first calculated the range of latent activations on a sample dataset, and then we set our random generator to produce values in the range: 0 (since we use ReLU activation functions) - 90th percentile of the real activation values. We used CORe50 NC and CORe50 NIC in these experiments.

The results are reported in Table 2. Surprisingly, even with random replay data (that we assume to be the worst degradation possible), negative replay is still able to perform better than no replay. Furthermore, the difference between original and generated data is minimal, thus proving that negative replay is tolerant in terms of data quality. Note that in both experiments using random data with negative replay performs way better than using generated data in a classical (positive) manner (PR-GD in previous figures). Comparisons in all the benchmarks of all the experiments (positive and negative replay with original, generated, and random data) are reported in Appendix E.

5. Related works

The use of negative examples to learn more discriminative class boundaries can be traced back to *one-class support vector machines (one-class SVM)* (Chen et al., 2001), where the data points belonging to the other classes in the training

Method	CORe50 NC	CORe50 NIC
No Replay	60.99 ± 0.49	52.71 ± 1.02
NR-OG	68.60 ± 1.38	67.93 ± 0.31
NR-GD	68.87 ± 0.88	61.46 ± 0.67
NR-RD	64.05 ± 0.71	58.85 ± 0.58

Table 2: Final accuracy on CORe50 NC and NIC using original (NR-OD), generated (NR-GD), and random (NR-RD) data with negative replay. The results with no replay are reported as references. Every experiment is averaged over 3 runs using different seeds and class orders.

set are used as negative examples. Malisiewicz et al. (2011) proposed using an ensemble of one-class SVMs instead of a single multi-class classifier. This approach operates in a scenario that is similar to the experiments on the CORe50 NIC benchmark, whose experiences contain only one class and all the replay data points (possibly belonging to many past encountered classes) are used as negative examples. The use of negative examples can also be seen as a kind of *contrastive learning* (Khosla et al., 2020), where negative examples are used to cluster embeddings of data points of the same class while moving away from embeddings of data from different classes.

Masking parts of a neural network have been experimented before in continual learning. Wortsman et al. (2020) masked the weights of a randomly initialized neural network to find a sub-network that yields good performance for a particular task. The loss masking proposed for ER using negative replay introduced in section 3.3 (without using any continual learning strategy) is similar to the continual learning method proposed by Masana et al. (2021). In that work, each feature can be used normally, masked (not used), or used only during forward (no modification of the related parameters during network update).

Generative replay for continual learning was first introduced by Shin et al. (2017) who proposed Deep Generative Replay (DGR). In that work, a generative adversarial network (GAN) (Goodfellow et al., 2014) is used as a generative model, showing promising results but only on simple datasets. The method

used a teacher-student framework, where the generative model resulting from the previous experience is used to train the current generative model. Wu et al. (2018) noted that generative replay almost completely shifts the problem of continual learning from the classifier to the generator. They proposed a GAN-based distillation approach to address the issue. However, these approaches lead to rapid degradation of the quality of the generated images for old tasks. To overcome this issue Ostapenko et al. (2019) proposed Dynamic Generative Memory (DGM), where a GAN architecture is used both to generate data and classify it. Moreover, the generative model used a combination of binary masks and network expansion, to maintain a fixed number of free parameters for every experience. All these works use GANs as generative models, but GANs are usually slow and complex to train, even in non-incremental scenarios. Kemker & Kanan (2018) proposed FearNet, a brain-inspired model that employs dual-memory storage (short and long term) with a transfer phase of information between the two memories in a consolidation phase inspired to mammalian sleep. Recently, Ayub & Wagner (2021) proposed a generative replay framework based on autoencoders and neural style transfer (Gatys et al., 2016) that showed interesting results even with high-dimensional data. However, that approach requires maintaining a generative model for every experience encountered so far, making it not scalable to long incremental sequences. Instead of generating raw images, van de Ven et al. (2020) proposed to generate internal features of the classifier through a Variational Autoencoder (VAE) (Kingma & Welling, 2014). This approach shares some similarities on how memory works in the human brain and with our proposed approach, showing significant results in continual learning scenarios with dozens of experiences. However, even this approach was not tested on high dimensional data and in scenarios with hundreds of experiences.

The crucial role of the dimensionality and the complexity of data on the quality of generation is evident in Zhai et al. (2019), where a simple generative model can effectively generate faithful results if trained continually on low-dimension data (e.g. the MNIST dataset LeCun (1998)), but it fails to generate acceptable results if the dimensionality of the data increases (e.g. using the

Flowers dataset (Nilsback & Zisserman, 2006)). Another example of this behavior is introduced by Mundt et al. (2019), where the Flower dataset, with image dimensionality of 256×256 pixels, is almost impossible to faithfully reconstruct if learned continually, even if the number of incremental experiences is low.

6. Conclusions

In this paper, we addressed the problem of continual learning with generative replay, focusing on the obstacles of generative replay in complex scenarios. Our experience confirms that incrementally training a generator over a long number of experiences with high dimensional data is a very challenging problem and remains an open issue. Therefore, instead of trying to design a better generative model, we focused on classifier training. We found that even inaccurate replay data can be useful to contrast the learning in isolation problem, especially in scenarios where only a limited number of classes is present in each experience. We called this approach negative replay since the replay data is used as negative examples when the model is trained with data from the current experience. We validated negative replay using complex continual learning scenarios, with high dimensional data and hundreds of incremental experiences. The results show that using negative replay largely improves classification performances w.r.t. using the generated data in a traditional fashion. We also investigated the impact of generated data quality, by considering the two extremes of using original data and random data for negative replay, and, surprisingly, we found that negative replay is effective even using random replay data.

Preliminary experiments have also been reported to show that negative replay can be easily applied to other continual learning strategies (besides AR1), and we believe that many other CL approaches may benefit from our proposal, especially when complex scenarios are addressed. Moreover, negative replay could be used in different scenarios, such as the pre-training phase of large models in order to make them more robust to noise or degraded data, to improve robustness against adversarial examples, or to address open set classification problems. Finally, our

replay experiments have been carried out by generating data in the latent space; in fact, as pointed out by many researchers this brings several advantages on complex high-dimensional problems (Hayes & Kanan, 2020; van de Ven et al., 2020; Pellegrini et al., 2020; Thandiackal et al., 2021); however, the evaluation of data quality in the latent space is more complex and further work will be necessary to better investigate the relationship between negative/positive replay and sample quality.

As a concluding remark, it is worth noting that dealing with imprecise replay data can be viewed as a biological feature since human's memory is far from being accurate, but is thought to be essential to consolidate learning (van de Ven et al., 2020), therefore investigating the role of negative replay-like mechanisms in biological learning could be an interesting research direction for computer and neuro-scientists.

References

Aljundi, R., Lin, M., Goujaud, B., & Bengio, Y. (2019). Gradient based sample selection for online continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 11816–11825). Curran Associates, Inc.

Ayub, A., & Wagner, A. R. (2021). EEC: Learning to encode and regenerate images for continual learning. In *International Conference on Learning Representations*.

Belouadah, E., & Popescu, A. (2019). Il2m: Class incremental learning with dual memory. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 583–592).

Belouadah, E., Popescu, A., & Kanellos, I. (2020). A comprehensive study of class incremental learning algorithms for visual tasks. *Neural Networks*, .

Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., & Alahari, K. (2018). End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 233–248).

Chaudhry, A., Dokania, P. K., Ajanthan, T., & Torr, P. H. S. (2018). Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 532–547).

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., & Ranzato, M. (2019). On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, .

Chen, Y., Zhou, X. S., & Huang, T. S. (2001). One-class svm for learning in image retrieval. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)* (pp. 34–37). IEEE volume 1.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., & Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (pp. 1–1). doi:10.1109/TPAMI.2021.3057446.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). doi:10.1109/CVPR.2009.5206848.

Dhar, P., Vikram Singh, R., Peng, K.-C., Wu, Z., & Chellappa, R. (2019). Learning without Memorizing. In *2019 IEEE Conference onComputer Vision and Pattern Recognition*.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2414–2423). doi:10.1109/CVPR.2016.265.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, *27*.

Hayes, T. L., & Kanan, C. (2020). Lifelong Machine Learning with Deep Streaming Linear Discriminant Analysis. In *CLVision Workshop at CVPR 2020* (pp. 1–15).

Hayes, T. L., Krishnan, G. P., Bazhenov, M., Siegelmann, H. T., Sejnowski, T. J., & Kanan, C. (2021). Replay in deep learning: Current approaches and missing biological elements. *arXiv preprint arXiv:2104.04132*, .

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Hempstalk, K., & Frank, E. (2008). Discriminating against new classes: One-class versus multi-class classification. In W. Wobcke, & M. Zhang (Eds.), *AI 2008: Advances in Artificial Intelligence* (pp. 325–336). Berlin, Heidelberg: Springer Berlin Heidelberg.

Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. In *Proceedings of the 5th International Conference on Learning Representations*.

Hou, S., Pan, X., Loy, C. C., Wang, Z., & Lin, D. (2019). Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 831–839).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, .

Huang, H., He, R., Sun, Z., Tan, T. et al. (2018). Introvae: Introspective variational autoencoders for photographic image synthesis. *Advances in neural information processing systems*, *31*.

Ioffe, S. (2017). Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *arXiv preprint arXiv:1702.03275*, .

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4401–4410).

Kemker, R., & Kanan, C. (2018). FearNet: Brain-Inspired Model for Incremental Learning. In *2018 International Conference on Learning Representations (ICLR)*.

Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., & Krishnan, D. (2020). Supervised contrastive learning. *Advances in Neural Information Processing Systems*, *33*, 18661–18673.

Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *PNAS*, *114*, 3521–3526.

LeCun, Y. (1998). The mnist database of handwritten digits.

Lesort, T., Caselles-Dupré, H., Garcia-Ortiz, M., Stoian, A., & Filliat, D. (2018). Generative Models from the perspective of Continual Learning. *Proceedings of the International Joint Conference on Neural Networks*, . doi:10.1109/IJCNN.2019.8851986.

Li, Z., & Hoiem, D. (2016). Learning without Forgetting. In *European Conference on Computer Vision* Springer (pp. 614–629).

Lomonaco, V., & Maltoni, D. (2017). CORe50: A New Dataset and Benchmark for Continuous Object Recognition. *CoRL*, .

Lomonaco, V., Maltoni, D., & Pellegrini, L. (2020). Rehearsal-Free Continual Learning over Small Non-I.I.D. Batches. *CVPR Workshop on Continual Learning for Computer Vision*, .

Malisiewicz, T., Gupta, A., & Efros, A. A. (2011). Ensemble of exemplar-svms for object detection and beyond. In *2011 International conference on computer vision* (pp. 89–96). IEEE.

Maltoni, D., & Lomonaco, V. (2019). Continuous Learning in Single-Incremental-Task Scenarios. *Neural Networks*, *116*, 56–73.

Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A. D., & van de Weijer, J. (2022). Class-incremental learning: Survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (pp. 1–20). doi:10.1109/TPAMI.2022.3213473.

Masana, M., Tuytelaars, T., & van de Weijer, J. (2021). Ternary feature masks: zero-forgetting for task-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 3570–3579).

McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of Learning and Motivation. Academic Press. doi:https://doi.org/10.1016/S0079-7421(08)60536-8.

Mundt, M., Majumder, S., Pliushch, I., Hong, Y. W., & Ramesh, V. (2019). Unified Probabilistic Deep Continual Learning through Generative Replay and Open Set Recognition. *arXiv*, . URL: http://arxiv.org/abs/1905.12019.

Nilsback, M.-E., & Zisserman, A. (2006). A visual vocabulary for flower classification. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (pp. 1447–1454). IEEE volume 2.

Odena, A., Olah, C., & Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning* (pp. 2642–2651). PMLR.

Ostapenko, O., Puscas, M., Klein, T., Jahnichen, P., & Nabi, M. (2019). Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 11321–11329).

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54–71. doi:10.1016/j.neunet.2019.01.012.

Pellegrini, L., Graffieti, G., Lomonaco, V., & Maltoni, D. (2020). Latent replay for real-time continual learning. In *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Pellegrini, L., Lomonaco, V., Graffieti, G., & Maltoni, D. (2021). Continual learning at the edge: Real-time training on smartphone devices. In *29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2021, Bruges, Belgium, October 6-8, 2021* (pp. 23–28). doi:10.14428/esann/2021.ES2021-136.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). iCaRL: Incremental Classifier and Representation Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shin, H., Lee, J. K., Kim, J., & Kim, J. (2017). Continual Learning with Deep Generative Replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 2990–2999). Curran Associates, Inc.

Sohn, K., Lee, H., & Yan, X. (2015). Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. volume 28.

Thandiackal, K., Portenier, T., Giovannini, A., Gabrani, M., & Goksel, O. (2021). Match what matters: Generative implicit feature replay for continual learning. *arXiv preprint arXiv:2106.05350*, .

van de Ven, G. M., Siegelmann, H. T., & Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, *11*. doi:10.1038/s41467-020-17866-2.

van de Ven, G. M., & Tolias, A. S. (2018). Three scenarios for continual learning. In *Continual Learning Workshop NeurIPS*.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., & Farhadi, A. (2020). Supermasks in superposition. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (pp. 15173–15184). Curran Associates, Inc. volume 33.

Wu, C., Herranz, L., Liu, X., van de Weijer, J., Raducanu, B. et al. (2018). Memory replay gans: Learning to generate new categories without forgetting. *Advances in Neural Information Processing Systems*, *31*, 5962–5972.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., & Fu, Y. (2019). Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 374–382).

Zenke, F., Poole, B., & Ganguli, S. (2017). Continual Learning Through Synaptic Intelligence. In *International Conference on Machine Learning* (pp. 3987–3995).

Zhai, M., Chen, L., Tung, F., He, J., Nawhal, M., & Mori, G. (2019). Lifelong gan: Continual learning for conditional image generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 2759–2768).

Appendix A. Details of the generative model implementation

We designed our generative model using different insights from previous works in the fields, bringing together different ideas and proposals. We extensively tested the generative model alone to find the better combination of building blocks that yield the best performance. Our design choices have been also influenced by the computation complexity since we aim to develop a (near) real-time system. This is a particularly hard constraint since many incremental generative replay methods are based on generative adversarial networks (GANs) (Goodfellow et al., 2014), which notably have long training phases and often suffer from instabilities due to the adversarial nature of the training procedure.

As discussed in the main text, we took inspiration from some state-of-the-art methods, trying to combine promising techniques and ideas from different sources. Taking inspiration from van de Ven et al. (2020) we use a Variational Autoencoder (VAE) model (Kingma & Welling, 2014), but unlike van de Ven et al. (2020) we opted for a conditional VAE (cVAE) configuration (Sohn et al., 2015). So, while in van de Ven et al. (2020) a mixture of Gaussian is used to sample latent vectors and soft labels are provided to the classifier itself, in our approach the latent vector is sampled from the normal distribution $\mathcal{N}(0, 1)$ and conditioned to the desired class. This results in a faster and less complicated sampling of a replay pattern. Moreover, as in van de Ven et al. (2020) we partially blend the encoder part of the generative model with the classifier model: both the networks share the same feature extractor f_ϕ. For the classifier, this branch is connected with the classification head c_ψ, while, for the generator, it is connected with some other layers that transform the feature into a latent vector z. The bifurcation is located in the latent replay layer. The resulting on-the-loop training of the generative model is consistent with brain structures

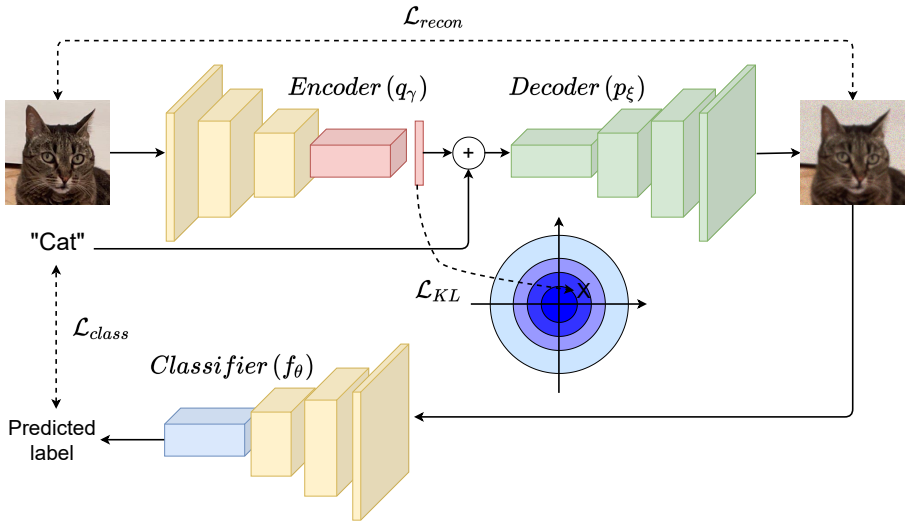

Figure A.6: A visual schema of the generative model training. Losses are represented by dashed arrows. The shared branch of the classifier and the encoder are depicted using the same color (yellow). The encoder and the classifier's additional layers are drawn in red and blue respectively.

and neuroscience's findings (van de Ven et al., 2020).

Since we use a cVAE, the objective for the generative model can be expressed as:

$$\gamma^*, \xi^* = \arg\min_{\gamma,\xi}[-\mathbb{E}_{z\sim q_\gamma(z|x_i^k)}[\log p_\xi(z|y_i^k)] + D_{KL}(q_\gamma(z|x_i^k)||p(z))], \tag{A.1}$$

where (x_i^k, y_i^k) are the data point and the label of the i-th pattern of the k-th experience, and the D_{KL} term represents the Kullback-Leibler divergence between the latent space distribution and the target distribution $p(z) = \mathcal{N}(0, 1)$.

The two terms of Equation A.1 determine two losses:

$$\mathcal{L}_{recon} = \|x_i^k - p_\xi(q_\gamma(x_i^k))\|_2^2 \tag{A.2}$$

$$\mathcal{L}_{KL} = D_{KL}(q_\gamma(x_i^k)||\mathcal{N}(0, 1)) \tag{A.3}$$

We also add another loss term, denoted as *classification loss*, which is similar to the classifier loss adopted in the AC-GAN model (Odena et al., 2017). The rationale is to guide the generative model to produce data that are not only

visually similar to the original ones (L2 loss) but that is also classified by the current classifier in the same way. Hence, we use f_Θ as "auxiliary" classifier, adding the following term to the generator's loss:

$$\mathcal{L}_{class} = -\log f_\Theta(y_i^k | p_\xi(q_\gamma(x_i^k)|y_i^k)), \tag{A.4}$$

which represents a typical negative log-likelihood classification loss. Note that the parameters Θ of the classifier are not trained in this phase, since only the generative model is updated. Overall, the generative model is trained using the following loss function:

$$\mathcal{L}_{GM} = \mathcal{L}_{recon} + \beta\mathcal{L}_{KL} + \eta\mathcal{L}_{class}, \tag{A.5}$$

where β is a hyper-parameter inspired to the β-VAE framework (Higgins et al., 2017), and η is a hyper-parameters that weights the importance of the classification loss.

A visual representation of generative model training is shown in Figure A.6.

To keep notation light, in the equations above the replay memory is not used, but it is trivial to include patterns from the replay memory, since there is no distinction in the generative model training procedure between current and replay data.

Note that the utilization of raw images is not mandatory for the method, and any intermediate (or latent representation) can be used, making our proposal compatible with latent replay methods (Pellegrini et al., 2020; van de Ven et al., 2020). In fact, in the case of latent replay, the data points x_i^k in the above equations can be simply substituted with $f_{\phi'}(x_i^k)$, where $f_{\phi'}$ is the set of feature extraction layers before the latent replay layer.

The blending of a part of the generative model into the classifier poses some difficulties in the training, especially regarding the balancing of the two models and how to train each of them without destructive inference on the other. After some initial experiments, we opted for blocking model parameters when the other model is trained. Detailed pseudo-code for the proposed negative generative replay strategy is provided in Algorithm 2.

Algorithm 2 Generative negative replay

$f_\Theta \leftarrow$ RandInit or PreTrained
$g_\Omega \leftarrow$ RandInit or PreTrained
$\mathcal{M}^x \leftarrow \emptyset, \;\; \mathcal{M}^y \leftarrow \emptyset$
$R =$ memory size
for each k from 1 tp N_E **do**
    if $k > 1$ **then**
        Sample $\{z_1, ..., z_R\} \sim \mathcal{N}(0, 1)$
        Sample $\{c_1, ..., c_R\} \sim \bigcup_{t=1}^{k-1} \mathcal{Y}_t$
        Block generator parameters (γ, ξ)
        Populate $\mathcal{M}_k^x = p_\xi(z_j|c_j)), \; j = \{1, ..., R\}$
        Populate $\mathcal{M}_k^y = \{c_1, ..., c_R\}$
    end if
    `# classifier training`
    $\psi' = \psi$
    Block generator parameters (γ, ξ)
    Unlock classifier parameters (ϕ, ψ)
    $\phi^*, \psi^* =$ Optimize$(f_\Theta, \mathcal{X}_k \cup \mathcal{M}_k^x, \mathcal{Y}_k \cup \mathcal{M}_k^y)$ using Equation 7
    WeightConsolidation$(\psi, \psi', \mathcal{Y}_k, \mathcal{M}_k^y)$ (see Algorithm 1)
    `# generator training`
    Block classifier parameters (ϕ, ψ)
    Unlock generator parameters (γ, ξ)
    $\gamma^*, \xi^* =$ Optimize$(g_\Omega, \mathcal{X}_k \cup \mathcal{M}_k^x, \mathcal{Y}_k \cup \mathcal{M}_k^y)$ using Equation A.5
end for

Appendix B. Validation of AR1 on ImageNet-1000

To validate the chosen AR1 algorithm we performed a test on a competitive benchmark on ImageNet-1000, following the NC benchmark proposed by Masana et al. (2022), which is composed of 25 experiences, each of them containing 40 classes. The benchmark is particularly challenging due to a large number of

classes (1,000), the incremental nature of the task (with 25 experiences), and the data dimensionality of 224×224 (as with ImageNet protocol).

With this experiment we want to assess the performance of AR1 in a complex continual learning scenario, validating the choice of AR1 as the main algorithm on which the majority of tests on negative replay are conducted. In this experiment, we tested AR1 against both regularization-based methods (Dhar et al., 2019; Kirkpatrick et al., 2017; Li & Hoiem, 2016) and replay-based approaches (Belouadah & Popescu, 2019; Castro et al., 2018; Chaudhry et al., 2018; Hou et al., 2019; Rebuffi et al., 2017; Wu et al., 2019). We use the same classifier (ResNet-18 (He et al., 2016)) and the same memory size for all the tested methods (20,000 patterns, 20 per class); for the regularization-based approaches, the replay is added as an additional mechanism.

For AR1, we trained the model with an SGD optimizer. For the first experience, we used an aggressive learning rate of 0.1 with momentum 0.9 and weight decay of 10^{-4}. We multiply the initial learning rate by 0.1 every 15 epochs. We trained the model for a total of 45 epochs, using a batch size of 128. For all the subsequent experiences we used SGD with a learning rate of $5 \cdot 10^{-3}$ for the feature extractor's parameters ϕ and $5 \cdot 10^{-2}$ for the classifier's parameters ψ. We trained the model for 32 epochs for each experience, employing a learning rate scheduler that decreases the learning rate as the number of experiences progresses. This was done to protect old knowledge against new knowledge when the former is more abundant than the latter. As in the first experience, the batch size was set to 128, composed of 92 patterns from the current experience and 36 randomly sampled (without replacement) from the replay memory.

The results are shown in Table B.3. Replay-based methods exhibit the best performance, with iCaRL and BiC exceeding a final accuracy of 30%. AR1 outperforms all the baselines (33.1%), demonstrating the validity of this approach also in difficult continual learning benchmarks. However, considering that top-1 ImageNet accuracy for a ResNet-18, when trained on the entire dataset, is

Method	Final Accuracy
Fine Tuning (Naive)	27.4
EWC-E (Kirkpatrick et al., 2017)	28.4
RWalk (Chaudhry et al., 2018)	24.9
LwM (Dhar et al., 2019)	17.7
LwF (Li & Hoiem, 2016)	19.8
iCaRL (Rebuffi et al., 2017)	30.2
EEIL (Castro et al., 2018)	25.1
LUCIR (Hou et al., 2019)	20.1
IL2M (Belouadah & Popescu, 2019)	29.7
BiC (Wu et al., 2019)	32.4
AR1 (Maltoni & Lomonaco, 2019)	**33.1**

Table B.3: Final accuracy on ImageNet-1000 following the benchmark of Masana et al. (2022) with 25 experiences composed of 40 classes each. For each method, a replay memory of 20,000 patterns is used (20 per class at the end of training). Results for other methods reported from Masana et al. (2022).

69.76%[2], even for the best methods the accuracy gap in the continual learning setup is very large. This suggests that continual learning, especially in complex scenarios with a large number of classes and high dimensional data, is far to be solved, and further research should be devoted to this field.

[2]Accuracy taken from the torchvision official page.

Appendix C. Classifier hyper-parameters

Appendix C.1. CORe50 NC

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	SI (Synaptic Intelligence) λ	$8 \cdot 10^5$
	SI Fisher matrix clip value	10^{-3}
	SI Fisher matrix multiplier	10^{-6}
1st experience	nr. epochs	4
	lr ϕ (feature extractor)	$3 \cdot 10^{-4}$
	lr ψ (classification head)	$3 \cdot 10^{-4}$
Following experiences	nr. epochs	4
	lr ϕ (feature extractor)	$3 \cdot 10^{-4}$
	lr ψ (classification head)	$3 \cdot 10^{-4}$

Table C.4: Hyper-parameters of the model trained with **no replay** using the **AR1 algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	SI (Synaptic Intelligence)	disabled
1st experience	nr. epochs	4
	lr ϕ (feature extractor)	$3 \cdot 10^{-2}$
	lr ψ (classification head)	$3 \cdot 10^{-2}$
Following experiences	nr. epochs	4
	lr ϕ (feature extractor)	$5 \cdot 10^{-5}$
	lr ψ (classification head)	$5 \cdot 10^{-4}$
	memory size	1,500
	replay pattern in the minibatch	14
	latent replay layer	conv5_4

Table C.5: Hyper-parameters of the model trained with **replay** (generative replay, random data, and real data), using the **AR1 algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
1st experience	nr. epochs	4
	learning rate	10^{-3}
Following experiences	nr. epochs	4
	learning rate	$3 \cdot 10^{-4}$

Table C.6: Hyper-parameters of the model trained with **no replay**, using the **ER algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
1st experience	nr. epochs	4
	learning rate	10^{-3}
Following experiences	nr. epochs	4
	learning rate	$3 \cdot 10^{-4}$
	memory size	1,500
	replay pattern in the minibatch	14
	latent replay layer	conv5_4

Table C.7: Hyper-parameters of the model trained with **replay** (generated and real data), using the **ER algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

Hyper-parameter	Value
optimizer	SGD
momentum	0.9
weight decay	10^{-4}
minibatch size	128
LwF α	0.1
temperature	2
nr. epochs	4
learning rate	$3 \cdot 10^{-4}$

Table C.8: Hyper-parameters of the model trained with **no replay**, using the **LwF algorithm**.

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	LwF α	0.1
	temperature	2
1st experience	nr. epochs	4
	learning rate	10^{-3}
Following experiences	nr. epochs	4
	learning rate	$3 \cdot 10^{-4}$
	memory size	1,500
	replay pattern in the minibatch	14
	latent replay layer	conv5_4

Table C.9: Hyper-parameters of the model trained with **replay** (generated and real data), using the **LwF algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

	Hyper-parameter	**Value**
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	SI (Synaptic Intelligence)	disabled
1st experience	nr. epochs	45
	lr ϕ (feature extractor)	10^{-1}
	lr ψ (classification head)	10^{-1}
	lr scheduler	lr $\cdot$ 0.1 every 15 epochs
Following experiences	nr. epochs	32
	lr ϕ (feature extractor)	$5 \cdot 10^{-3}$
	lr ψ (classification head)	$5 \cdot 10^{-2}$
	lr scheduler	see Equation C.1

Table C.10: Hyper-parameters of the model trained with **no replay**, using the **AR1 algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	SI (Synaptic Intelligence)	disabled
1st experience	nr. epochs	45
	lr ϕ (feature extractor)	10^{-1}
	lr ψ (classification head)	10^{-1}
	lr scheduler	lr $\cdot$ 0.1 every 15 epochs
Following experiences	nr. epochs	32
	lr ϕ (feature extractor)	$5 \cdot 10^{-3}$
	lr ψ (classification head)	$5 \cdot 10^{-2}$
	lr scheduler	see Equation C.1
	memory size	20,000
	replay pattern in the minibatch	36
	latent replay layer	layer4 (4th resnet block)

Table C.11: Hyper-parameters of the model trained with **replay** (generative replay and real data), using the **AR1 algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

Due to the complexity of the ImageNet-1000 scenario, we found it useful to use a learning rate scheduler that decreases the learning rate as the number of experiences progresses. The scheduler can be formalized as:

$$\text{lr} = \text{lr}_{init} \cdot \left(-\frac{0.9}{1 + e^{-1.5i+8}} + 1 \right), \tag{C.1}$$

where i indicates the index of the current experience.

	Hyper-parameter	**Value**
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	SI (Synaptic Intelligence) λ	$2.3 \cdot 10^{6}$
	SI Fisher matrix clip value	10^{-3}
	SI Fisher matrix multiplier	$2 \cdot 10^{-5}$
1st experience	nr. epochs	4
	lr ϕ (feature extractor)	10^{-3}
	lr ψ (classification head)	10^{-3}
Following experiences	nr. epochs	4
	lr ϕ (feature extractor)	10^{-4}
	lr ψ (classification head)	10^{-3}

Table C.12: Hyper-parameters of the model trained with **no replay**, using the **AR1 algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

	Hyper-parameter	Value
Common	optimizer	SGD
	momentum	0.9
	weight decay	10^{-4}
	minibatch size	128
	SI (Synaptic Intelligence)	disabled
1st experience	nr. epochs	4
	lr ϕ (feature extractor)	10^{-3}
	lr ψ (classification head)	10^{-3}
Following experiences	nr. epochs	4
	lr ϕ (feature extractor)	10^{-4}
	lr ψ (classification head)	10^{-3}
	memory size	300 (N/A for random data)
	replay pattern in the minibatch	64 (21 for random data)
	latent replay layer	conv5_4

Table C.13: Hyper-parameters of the model trained with **replay** (generative replay, random data, and real data), using the **AR1 algorithm**. Common hyper-parameters are the same for each experience, 1st experience hyper-parameters are used in the first experience, the following experience hyper-parameters are used in all the following experiences.

Appendix C.4. On the amount of replay data in the minibatch

The amount of replay data included in each minibatch has a direct impact on the performance of the continual learning strategy adopted. We observed that the optimal value changes with the quality of the replay data and that a large amount of degraded replay data in each minibatch may decrease disruptively the performance of the model.

We compared different original/replay proportions, finding that when using real replay data, the model is not much sensitive to the amount of replay data in the minibatch and different proportions work well: we empirically noticed a peak of performance around a 50-50 split. Using generated (degraded) or random

data is quite different. We noticed that if the data used is highly degraded the maximum gain in performance is when 10-30% replay data are added. Exceeding 30% usually leads to a degradation of performance, and if the amount of replay data is still higher (depending on the replay data quality) the accuracy of the model can be lower than not using replay data.

Appendix D. Generative model hyper-parameters

Appendix D.1. CORe50 NC

	Hyper-parameter	**Value**
Common	optimizer	Adam
	betas	0.9 - 0.999
	weight decay	0
	minibatch size	128
	latent space dim.	100
	β	0.1
	η	0.01
	lr	$2 \cdot 10^{-3}$
	lr scheduler	None
	nr. epochs	4
Following experiences	replay patterns in the minibatch	27

Table D.14: Hyper-parameters of the generative model trained on CORe50 NC. Common hyper-parameters are the same for each experience, while following experience hyper-parameters are used in all the experiences except the first one.

	Hyper-parameter	**Value**
Common	optimizer	SGD
	momentum	0
	weight decay	0
	minibatch size	128
	latent space dim.	100
	β	0.25
	η	0.01
	lr	1
	lr scheduler	see Equation C.1
	nr. epochs	32
Following experiences	replay patterns in the minibatch	36

Table D.15: Hyper-parameters of the generative model trained on ImageNet-1000 NC. Common hyper-parameters are the same for each experience, while following experience hyper-parameters are used in all the experiences except the first one.

	Hyper-parameter	**Value**
Common	optimizer	Adam
	betas	0.9 - 0.999
	weight decay	0
	minibatch size	128
	latent space dim.	100
	β	0.1
	η	0.01
	lr	$2 \cdot 10^{-3}$
	lr scheduler	None
	nr. epochs	4
Following experiences	replay patterns in the minibatch	64

Table D.16: Hyper-parameters of the generative model trained on CORe50 NIC. Common hyper-parameters are the same for each experience, while following experience hyper-parameters are used in all the experiences except the first one.

Appendix E. Additional plots

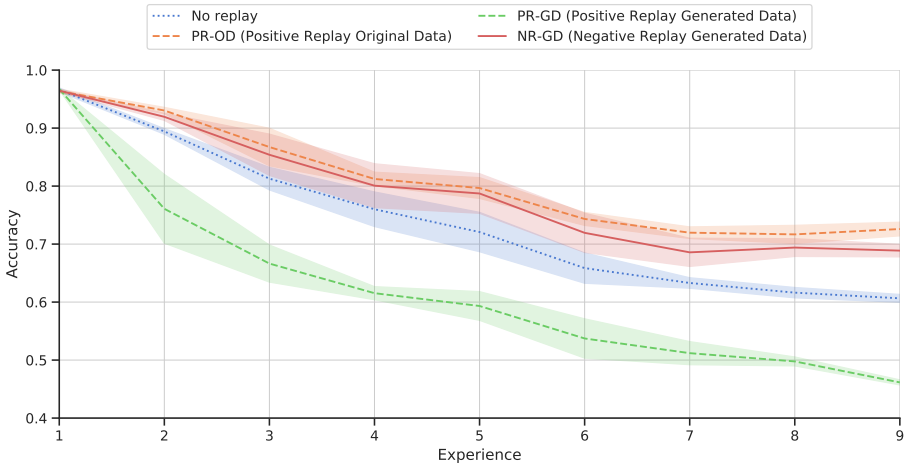

Figure E.7: Overall accuracy on the CORe50 NC scenario, using a growing test set. After each experience, the model was evaluated using a test composed of only data belonging to the classes seen so far, similar to the benchmark proposed by Masana et al. (2020). Every experiment is averaged over 3 runs, with different seeds and class order. The standard deviation is reported in light colors. Better viewed if zoomed on a computer monitor.

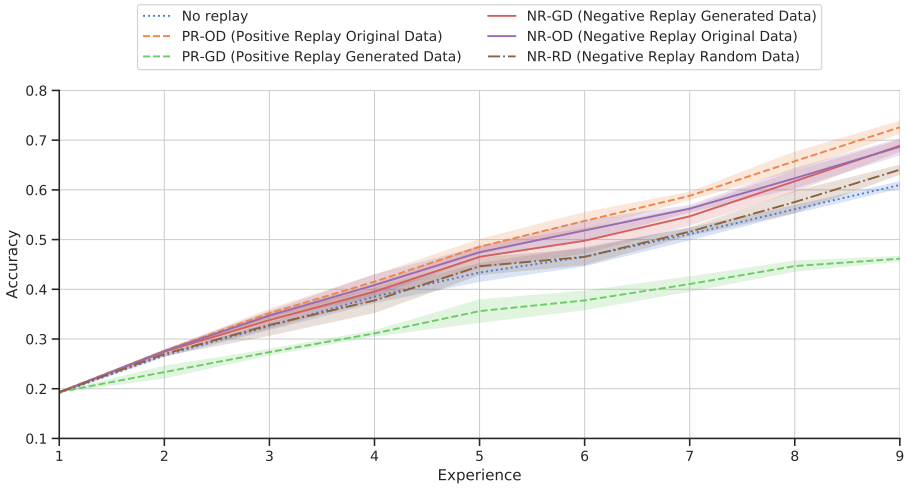

Figure E.8: Overall accuracy on the CORe50 NC scenario for all the experiments performed in this work (included random data and negative replay with original data). After each experience, the model was evaluated using the cumulative test set as proposed by Lomonaco & Maltoni (2017). Every experiment is averaged over 3 runs, with different seeds and class order. The standard deviation is reported in light colors. better viewed if zoomed on a computer monitor.

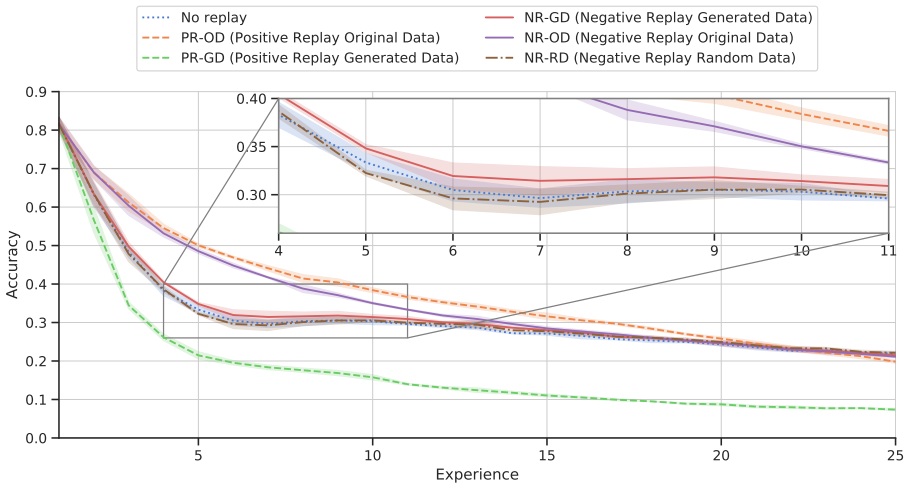

Figure E.9: Overall accuracy on the ImageNet-1000 NC scenario for all the experiments performed in this work (included random data and negative replay with original data). After each experience, the model was evaluated using the whole test set as proposed by Masana et al. (2022). Every experiment is averaged over 3 runs, with different seeds and class order. The standard deviation is reported in light colors. better viewed if zoomed on a computer monitor.

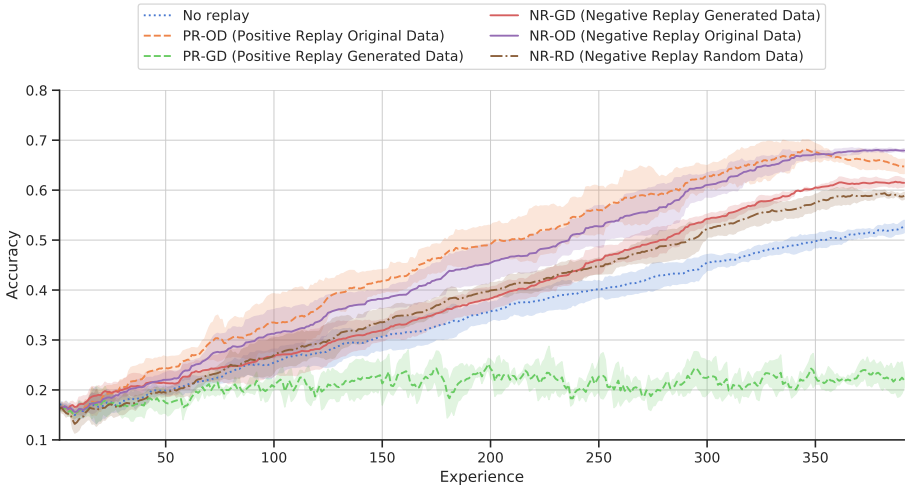

Figure E.10: Overall accuracy on the CORe50 NIC scenario for all the experiments performed in this work (included random data and negative replay with original data). After each experience, the model was evaluated using the cumulative test set as proposed by Lomonaco & Maltoni (2017). Every experiment is averaged over 3 runs, with different seeds and class order. The standard deviation is reported in light colors. better viewed if zoomed on a computer monitor.