Deep online classification using pseudo-generative models

Andrey Besedin, Pierre Blanchart, Michel Crucianu, Marin Ferecatu

Computer Vision and Image Understanding
journal homepage: www.elsevier.com

Deep Online Classification Using Pseudo-Generative Models

Andrey Besedin[a,b], Pierre Blanchart[a], Michel Crucianu[b], Marin Ferecatu[b]

[a] *CEA, LIST, Laboratoire d'Analyse de Donnees et Intelligence des Systemes, Digiteo Labs Saclay, 91191, Gif-sur-Yvette Cedex, France*
[b] *Centre d'Etudes et de Recherche en Informatique et Communications, Le CNAM, 292 rue Saint-Martin, 75003, Paris, France*

ABSTRACT

In this work we propose a new deep learning based approach for online classification on streams of high-dimensional data. While requiring very little historical data storage, our approach is able to alleviate catastrophic forgetting in the scenario of continual learning with no assumption on the stationarity of the data in the stream. To make up for the absence of historical data, we propose a new generative autoencoder endowed with an auxiliary loss function that ensures fast task-sensitive convergence. To evaluate our approach we perform experiments on two well-known image datasets, MNIST and LSUN, in a continuous streaming mode. We extend the experiments to a large multi-class synthetic dataset that allows to check the performance of our method in more challenging settings with up to 1000 distinct classes. Our approach is able to perform classification on dynamic data streams with an accuracy close to the results obtained in the offline classification setup where all the data are available for the full duration of training. In addition, we demonstrate the ability of our method to adapt to unseen data classes and new instances of already known data categories, while avoiding catastrophic forgetting of previously acquired knowledge.

1. Introduction

In recent years, methods based on Deep Learning (DL) have become state of the art in numerous applications, such as image and signal classification (Krizhevsky et al. (2012)), object detection (Sermanet et al. (2013)), recognition (Yu et al. (2019)), and segmentation (He et al. (2017)), natural language processing (Sutskever et al. (2014)), (Collobert et al. (2011)), privacy protection (Yu et al. (2017)), and many others. Despite their popularity and efficiency on high-dimensional data of significant structural complexity, most of the currently existing deep learning approaches are aiming to solve offline learning problems where all the data are constantly available during training. With the fast development of robotics, social networks, and personalized gadgets collecting user information, the demand for real-time processing of huge amounts of information is rapidly growing. Recently, online learning scenarios, where data arrive continuously in large quantities and have to be integrated into the learning system in real-time, start to receive significantly more attention from the Machine Learning (ML) community.

Performing Online Deep Learning, and specifically online classification using Neural Networks (NN), is not straightforward and requires special attention. The main difficulty is that NN-based classifiers usually rely on the assumption that the sequence of data batches used during training is stationary, or in other words, that the distribution of data classes is the same for all batches. Sometimes the term i.i.d. is equivalently used in the literature to express the fact that the batches (1) are randomly sampled from the training data (implying independence) and (2) are sampled from the same distribution (identically distributed). Because back-propagation tends to re-

inforce the classes present in the current batch, when this i.i.d. assumption does not hold neural networks tend to forget the concepts that are temporarily not available. In literature this phenomenon is known as *catastrophic forgetting* (McCloskey and Cohen (1989)).

On the other hand, learning in neural networks is a slow process that often requires several training epochs through the full dataset. This is unfeasible in online classification training where the model usually has to be updated in real-time while new data arrive. Another problem is due to the non-stationarity of the stream: new classes are appearing, as well as new modalities of previously seen classes. This induces catastrophic forgetting when learning only on new data. A straightforward solution to this problem is rehearsal (Robins (1995)) that consists in storing all (or at least a significant part of) the historical data together with newly arriving stream data and retraining the classifier on all the available data every time the model has to be updated. However, if the system is required to learn on fast data streams and keep acquiring new knowledge for a long period of time, or has to perform life-long learning, such an approach fails to scale to the size of the problem.

To avoid the difficulties related to historical data storage, a recent popular line of research is the use of generative models to approximate the statistical distribution of the source (Parisi et al. (2018); Shin et al. (2017); Kamra et al. (2017)). In our recent work (Besedin et al. (2017, 2018)), we employed GANs as generative models and trained them to produce images that are visually close to the input source. Each time a data class disappears from the input stream, we use the previously learned GANs (one for each class) to generate new samples for the missing classes in such a way that all classes (past and present) are equally balanced in the current learning batch. While the instability and the convergence of the training process in GANs have been addressed by several recent works (Salimans et al. (2016); Arjovsky et al. (2017); Miyato et al. (2018)), training GANs that approximate well the distribution of the original data is a complex task, typically slow and very data-intensive. Indeed, in Besedin et al. (2018) we demonstrated that to be able to efficiently replace the original data for the classification task on the LSUN dataset, each GAN required several millions of training updates for each class. This is a strong limitation with continuous streams where one has to learn on the fly and from potentially very unbalanced classes.

In this paper we propose a new framework that improves in several ways on our previous work, solving the above mentioned problems. We focus on the general case of non-i.i.d. streams of unordered high-dimensional data and we formalize the problem as real-time continual learning without forgetting. In the absence of historical data, we advocate the use of autoencoders as pseudo-generative models (instead of GANs) to deal with the absent classes. By high-dimensional data we understand objects (coming through the stream) that are described by a large number of variables. In this work we are mainly interested in images, a type of data on which non-Deep Learning methods do not perform so well. However, we do not use the images directly, but rather the convolutional features obtained from a pre-trained network (see Sec. 4.1).

The paper is organized as follows: in Sec. 2 we discuss the main challenges of online learning on data streams, review existing methods that address this problem and position our proposal with respect to them. In Sec. 3 we present our proposal: we define the problem of continual learning from dynamic data streams, discuss the challenges that motivate our approach, describe the techniques that we apply for online learning and justify the use of pseudo-generative models to replace historical data. In Sec. 4 we assess the performance of our online learning approach on three different datasets and analyze its behavior, while in Sec. 5 we discuss perspectives and future work.

2. Related work

2.1. Main challenges for Deep Learning on Dynamic Data

In this section, we outline the main challenges one has to face in online classification using Deep Learning based models and discuss existing methods that address these challenges.

Though the data distribution might not necessarily change with time in an online classification scenario, the way one

has to store and process the data significantly changes compared to the static learning case, resulting in new problems to solve. Until recently, the most widely used methods for classification on data streams were not based on Deep Learning and mainly included Hoeffding trees (Domingos and Hulten (2000)), Bayesian trees (Seidl et al. (2009)), Support Vector Machines (Rai et al. (2009)) and ensemble methods (Oza (2005)). A comparative overview of these methods can be found in Nguyen et al. (2015) with the main conclusion that, even though they allow real-time testing, are able to efficiently handle concept drift and do not face memory issues (Webb et al. (2016)), non-DL methods do not perform very well on complex high-dimensional data such as images.

In comparison, DL methods are efficiently handling complex classification tasks on high-dimensional data with up to a few thousand classes. However, training DL models on non-stationary stream data with a growing number of classes generally leads to catastrophic forgetting. The knowledge encoded in the neural connections is gradually overwritten by new information in the absence of data reinforcing previous knowledge, *i.e.* data corresponding to previously learned classes or to different modes of the current classes. In the case of re-training only on the newly received data the model gradually learns to classify the data into the new classes without preserving its ability to classify correctly into classes now absent from the stream.

The re-training problem is exacerbated by the fact that the amount of data from each class that appears during a time interval is not constant. In an unordered stream, which is the general setting for stream data classification, the presence of a given class in a given time interval is not even guaranteed.

Many methods have recently been proposed to solve or alleviate the problem of catastrophic forgetting. The existing methods can be grouped into three categories: regularization-based, adaptive architectures and dual-memory systems. To better understand the specificity of these approaches, their advantages and drawbacks, and find where to situate the method we propose in this paper, we discuss them below.

2.2. *Regularization-based methods*

Regularization-based methods aim to control the way the parameters of the model are updated during training in order to limit the variation of important connection weights and thus avoid erasing the information from the old tasks.

Li and Hoiem (2017) put forward Learning without Forgetting (LwF), a method that consists in initializing a separate set of parameters for each new task, together with a large pool of parameters shared across all the tasks. Shared parameters are updated very slowly to avoid changing model performance for previously learned tasks, while task-specific parameters get updated with no restrictions. The drawback of such a method is its computational complexity that grows linearly with the number of tasks. In addition, the proposed approach is highly dependent on the similarities among tasks and requires supplementary data storage for each task.

The Elastic Weight Consolidation Kirkpatrick et al. (2017) consists in introducing a supplementary quadratic penalty for the difference between the updated model and its historical version, trained for previous tasks. The penalty slows down updates for the weights relevant to the old tasks. The authors compute the Fisher information matrix and use it to perform the diagonal weighting over the parameters of the learned tasks. This procedure can only be performed offline, which makes the proposed approach not applicable to online learning problems.

Zenke et al. (2017) present an online learning method that adapts to stream non-stationarity by dubbing each network connection with a biologically inspired synapse in which task-relevant information is accumulated over time and then exploited when updating network weights to avoid forgetting.

In Hou et al. (2019) and Zhang et al. (2019) the authors propose to use knowledge distillation (Hinton et al. (2015)) that helps limiting the information asymmetry between the old and the new classes when learning from data streams. However, the proposed method only suits the incremental learning scenario and requires training each new task until convergence. Although it outperforms most regularization-based approaches, knowledge distillation remains less efficient than joint training

to prevent catastrophic forgetting.

While these regularization approaches avoid forgetting in specific setups, they are usually limited to batch learning and introduce a very sensitive trade-off between knowledge consolidation and the incorporation of new information.

2.3. Dynamic architectures

Dynamic architecture methods are mainly based on allocating supplementary resources to the learning system on demand, *e.g.* when new data types or classes are added.

Rusu et al. (2016) proposed Progressive Neural Networks, a method that consists in adding a new neural network with a predefined architecture for each new task. This approach showed a good capacity to avoid forgetting and to perform efficient knowledge transfer in multi-task scenarios. However, model complexity and memory requirements increase dramatically with the growing number of tasks.

Fernando et al. (2017) put forward another method, PathNet, that introduces agents controlling the information propagation inside the network. The agents are trained to choose the optimal paths inside the network and allocate sub-networks for every specific task. Though very efficient for knowledge transfer, the described experimental setup requires to learn each new task until convergence which is unfeasible in online learning.

Yoon et al. (2018) proposed Dynamically Expanding Networks, an approach that selectively retrains parts of the old model while modifying the network by adding new neurons or removing old ones. The choices of architecture modifications are made with the help of group sparse regularization.

All these methods employing dynamic architectures have a further major drawback: at testing time such models need to know which task they are currently solving and thus testing cannot be made in a completely unsupervised manner.

2.4. Dual-memory approaches

Dual-memory approaches rely on the use of external models or supplementary storage units to store and reuse previously acquired knowledge.

In Calandra et al. (2012), Besedin et al. (2017) and Shin et al. (2017) the authors employ generative models – respectively a Deep Belief Network for each class, a GAN for each class, and a GAN for each task – to generate representations of the historical data and reuse them instead of the original data when needed. This helps to regularize training and avoid forgetting of missing data classes. These methods have shown stable performance on the MNIST dataset, but less so on more complex datasets. Also, their scalability to a large number of classes is questionable due to the fact that they train one generative model per class/task, which may be a problem with complex generative models such as GANs that require heavy hyperparameter adjustment.

Kemker and Kanan (2017) propose to combine a buffer keeping the most recent stream samples as the short-term memory and symmetric generative autoencoders as the long-term memory. Codes resulting from the encoder are passed through the softmax and compared to the one-hot vector of corresponding class labels. By doing so, the code space gets partitioned into class-specific regions, over which the authors fit a Gaussian Mixture Model with as many mixture components as classes. Synthetic codes of a specific class can then be sampled from that distribution and passed through the decoder to generate replacement data of the missing class. A major drawback of such an approach is that the underlying latent codes distribution for each class has to be explicitly computed. Limitations spring both from the low modelling power of GMMs (mixtures of Gaussians do not yield good models in high-dimensional spaces, such as the latent code space) and from the updating abilities of such a model which, as a statistical model, is subject to local minima during optimization and low reactivity to a small amount of new data. As such, this method is better adapted to the incremental learning scenario – the one evaluated by the authors – where data classes, once learned, never reappear in the stream. Moreover, due to the specific classification loss, the size of the encoded features has to be fixed, so this approach can not be applied to problems where the total number of classes is unknown.

While all the described methods address the problem of catastrophic forgetting in Neural Networks, none of them fully considers the general case of continual learning on an unordered stream with a high number of classes. In Parisi et al. (2018), the authors arrive at the same conclusion that the drawbacks of existing methods or their limitation to a specific setting prevent them from being applicable in the general case.

In this paper we place our proposal in the category of dual-memory approaches. But instead of using a generative model, such as a GAN, to supply (when needed) samples similar to the historical ones, we store enough historical data compressed into very low dimensional representations (or codes) by using a type of autoencoders specifically enhanced for this task. We train the main classifier with a mix of newly arrived stream data and historical data reconstructed from randomly selected stored representations. In the following we call this approach of recalling historical data a "pseudo-generative model". More specifically, the contributions of the paper are as follows:

1. We propose the use of autoencoders as pseudo-generative models to avoid catastrophic forgetting while keeping memory requirements low. Autoencoders project input samples to a low-dimensional feature space by minimizing the reconstruction error. Instead of learning a generative model that tries to mimic a high-dimensional statistical distribution, which is known to be a difficult problem, we store projections (via the autoencoder) of samples from the input stream, in a number that is small but sufficient to alleviate the loss of memory. When training on a new batch of arriving data, we add reconstructed samples from the missing classes by using part of the stored projections.

2. We put forward a new loss function for the autoencoder that takes into account our specific goal: the reconstructed samples should behave well with respect to reinforcing the classes already learned and not merely minimize the reconstruction error. Following this logic, the loss function we propose takes into account the error that the generated samples produce during classification.

3. We argue that for very complex data, such as high-resolution natural images, training can be accelerated without loss in classification accuracy by passing data through a pretrained feature extractor, such as Resnet152 (He et al. (2016)) and further training of both classifier and pseudo-generative models on the extracted features. This step allows us to increase the training speed and makes possible the application of our method to massive data streams.

We test our framework on the well-known MNIST[1] benchmark and on the much larger LSUN[2] dataset ($3 \cdot 10^6$ images in 30 classes). Finally, we confirm the scalability of our approach to a much larger number of classes by evaluating it on a synthetic dataset of $15 \cdot 10^6$ data samples with up to 10^3 separate classes. We explain in detail our proposal in the next section.

3. Pseudo-generative models for online classification without forgetting

While the domain of continual learning is quickly expanding, there is still little agreement on the formalization of the problems it addresses. In this section we point out the main requirements for a system that aims at "human-like" continual learning and, to make the language precise, we give a few formal definitions. Based on these, we introduce the experimental scenario studied in this paper and describe our approach to handle it.

3.1. Problem statement and stream simulation

In most of the works discussed in Sec. 2, the authors place themselves in an incremental learning scenario. In this case, a task consists in adapting a classifier to a new domain (*e.g.*, new image classes) using training data corresponding to the new domain and a pre-trained version of the classifier in a related source domain. The aim is to train a model on a sequence of related tasks T_i that come each with a separate training dataset D_i. No assumption is made, though, as to the distributions of datasets D_i being identical. The goal of incremental learning is to optimize the performance of the model on each new task while preserving its ability to solve previous tasks. In this setting, the learning system usually requires a separate model or

[1] http://yann.lecun.com/exdb/mnist/

[2] https://www.yf.io/p/lsun

at least a separate output layer per task. As a consequence, for a given test data we need to know to which task it is related, *i.e.* to which subset of classes it belongs to. Moreover, in incremental learning each task is trained until convergence, excluding further modification/improvement of older tasks, even if we observe in the stream new data related to these tasks.

In this work we consider the more difficult problem of continual learning from data streams. We define a data stream S as the result of sequential data acquisition from one or multiple sources, batch after batch. We suppose that data acquisition is performed by standardized sensors so that the data samples are all of the same nature and format. Since we consider classification problems, we assume that during the training phase all data samples come with corresponding class labels, while for testing no labels are available. We also suppose that each data sample belongs to a single class, *i.e.* that it is associated with one single label. We consider that small batches of data are continuously sent to the learning system from the dynamically changing environment. Let us denote by $\mathcal{D}$ the set of labeled samples from which we simulate the stream. We denote by $\mathcal{E} = \{l_1, \ldots, l_N\}$ the set of labels of all the available classes. To simulate the stream S we divide it into non-i.i.d. (in terms of class label distribution) time intervals I_t, each containing a number of distinct class labels $N_t < N$. We denote by $\mathcal{E}^t$ the set of N_t distinct labels that form I_t:

$$\mathcal{E}^t = \left\{l_1^{(t)}, \ldots, l_{N_t}^{(t)}\right\}$$

where the $l_i^{(t)}$ are the distinct class labels. The set of classes already seen by the system is referred to as $\mathcal{E}^{hist}$.

We simulate streams by sampling data according to their class label distribution. We use the notation

$$D_{\mathcal{E}} = \{(X, y) \in \mathcal{D} \,|\, y \in \mathcal{E}\}$$

to refer to the subset of data with the corresponding class label in $\mathcal{E}$. For each time interval I_t, we sample uniformly class labels from $\mathcal{E}^t$ and form associated data subsets $D_{\mathcal{E}^t}$. We simulate a batch b_s of data arriving from the streaming environment at time t by sampling uniformly the elements of b_s from $D_{\mathcal{E}^t}$.

The main challenge of learning from streamed data is to be able to continuously learn from the new data while preserving the knowledge acquired on past data. With our notations, for each interval I_t we want to optimize the classification accuracy of the model on the data classes $\mathcal{E}^t$ present in that interval while preserving the model performance on historical classes $\mathcal{E}^{hist}$.

After defining the notations and the stream model we employ, in the following sections we put forward our approach supporting efficient learning from dynamic data streams.

3.2. Online learning with the help of generative models

As described in Sec. 2, one way to avoid forgetting in online learning is to use supplementary storage modules or models in order to "replay" historical data or to generate replacement data when necessary (Sec. 2.4). To do so, a supplementary component that serves for long-term memorization is required. A typical approach is to use a rehearsal mechanism that stores the most relevant instances of previously seen classes in a buffer. It is then employed to supply the classification model with historical data when needed. The main limitation of this approach is that training DL models on complex high-dimensional data requires a large amount of data samples per class. But storing and reusing large amounts of data is infeasible in a massive real-time stream scenario due to storage limitations and high re-training cost.

Instead of using the historical data directly, we rather approximate its distribution by a generative model that allows to sample labeled instances similar to real data at any moment of the stream. In our experimental scenario we train two separate models: a NN classifier model C and a generative model G. The classifier C receives data samples as input and outputs the probabilities of the input belonging to one of the already seen classes. Each time a new data class (not already in $\mathcal{E}^{hist}$) appears in the stream, a new neuron is added to the output layer so that the latter constantly contains as many neurons as there are classes in the current state of the stream environment. The generator G is trained to produce replacement data approximating the distribution of the original data. It receives a latent code as input and outputs a tensor of the shape of the data.

One of the key goals of our approach is to remove the need for storing excessive amounts of historical data. However, ex-

periments in Sec. 4.2.3 show that keeping a small number of real data samples per class is still essential to avoid progressive drifts in the distribution of generated data. Thus, to perform online learning we initialize a small (compared to the size of the original dataset) buffer B^{hist} that serves as short-term memory. It is a set of cells, one for each class already seen by the system, each storing a small portion (less than 1%) of the most recent real data samples of the corresponding class.

In most real life applications that require online learning we are likely to have prior access to at least a small dataset representing our global task. The natural approach in this case is to make use of this dataset to pretrain both models, C and G, prior to learning from the stream. In our experimental setup (Sec. 4.1) we consider a similar case: we start with an initial training subset $D^{init} = D_{\{l_1,\ldots,l_{N/2}\}}$ containing all the training data associated with half the number of classes in the full data environment. We pretrain both C and G on D^{init}, then pursue with online training on the stream. The D^{init} subset is also used for fine-tuning different parameters by cross-validation (see Sec. 4.2).

During the online learning phase, for each time interval I_t the system receives several batches b^{stream} of stream data. For each batch b^{stream}, we sample a batch b^{old} of the same size from B^{hist} and generate K batches using the generator G, where K is a predefined parameter (generated-to-real data ratio) that controls the preservation of the already learned classes. The resulting batches are then joined together and used to perform a gradient descent training step for both G and C. The buffer B^{hist} is then updated with the elements in b^{stream}. If a new class is introduced in b^{stream}, a new cell is added to B^{hist}. Otherwise, the oldest samples from the corresponding cells in B^{hist} are replaced by the elements in b^{stream}. The online learning procedure described above is summarized in Alg. 1.

Require: D : full data environment
Require: $\mathcal{E} = \{l_1, \ldots, l_N\}$: set of class labels in D
Require: $S = \{I_t, t = 1, 2, \ldots\}$: data stream
Require: s : random data sampler
Require: K : generated-to-real data ratio
Require: bs : batch size
  Initialize $\mathcal{E}^{init} = \{l_1, \ldots, l_{N/2}\}$
  Initialize $D^{init} = D_{\mathcal{E}^{init}}$
  Initialize B^{hist} from D^{init}
  Initialize C and G
  Train C and G on D^{init} offline
  for I_t in S **do**
    for b^{stream} in I_t **do**
      $b^{old} \Leftarrow s(B^{hist}, bs)$ – get historical data batch
      $b^{gen} \Leftarrow s(G, K \times bs)$ – get generated batch
      $b \Leftarrow$ concatenate$(b^{stream}, b^{old}, b^{gen})$
      Update C and G on b
      Update B^{hist} from b^{stream}
    end for
  end for

Algorithm 1: Online learning from a data stream. For each time interval I_t the classifier C and the generative model G are updated with new batches of samples containing a mix of newly arrived data, generated data and historical data. The sampler $s(S, n)$ produces n data points from the source S with a uniform distribution over the set of classes already seen by the system.

3.3. Training generative models on data streams

In spite of the popularity of generative models, there are relatively few studies on the usability of the generated representations for learning tasks. For instance, in natural image synthesis, the authors evaluate the generative models by their ability to fool a human expert. It is questionable whether a similar visual assessment of the usability of generated data in place of real data for a given learning task is relevant and practical.

A quantitative measure, the Inception Score, was proposed by Salimans et al. (2016). Their evaluation method consists in applying an Inception model pre-trained on ImageNet to a set of generated samples. The resulting conditional label distribution $p(y|x)$ of the samples should be close to uniform (*i.e.* have high entropy), while for any given input sample the classification output vector should have a peak (low entropy). But Bar-

ratt and Sharma (2018) show that the Inception Score is often misleading for generative models trained on datasets other than ImageNet. In addition, such an evaluation only allows to assess general characteristics of the distribution of generated samples.

In our study we use generative models for the specific task of online classification. Thus, we rather have to evaluate the generator by measuring to what extent the generated samples are able to replace well the real data when learning the stream classifier. The goal is to make sure that, with respect to the classifier, the generated data is "similar" to the real data. The working scenario (multi-class online classification from streams) imposes a number of constraints on the generative model:

- **Reactivity**: the model should be fast enough to process batches of data from the stream in real-time.
- **Conditional sampling** ability: the classification task requires the generative model to be able to approximate multi-class data distributions and perform label-conditioned sampling.
- **Scalability**: with continual learning it is impossible to know in advance the total number of classes the system will have to learn. Thus, in order to scale to problems with a large number of classes, the model should grow slower than linearly with the number of data classes.
- **Learning efficiency**: due to the online nature of the learning procedure, only one pass through each stream sample is allowed. Therefore, models should be able to rapidly incorporate new knowledge.
- **Protection from forgetting**: we use generative models to avoid catastrophic forgetting in a classifier, but the generative model itself should also be protected from forgetting when some classes are not constantly present in the stream.

While all the above characteristics are important, the first three (learning reactivity, conditional sampling and scalability) translate into architectural constraints on the model. To ensure learning reactivity, we train models with shallow architectures and employ extracted feature representations of complex image data (see Sec. 4.1).

Generating data together with the corresponding labels is not straightforward for most of the existing generative approaches, while being a compulsory condition for the online classification task. Our preliminary experiments showed that while having good capacity to approximate data distributions and generate realistic samples, currently existing generative models based on adversarial training (Goodfellow et al. (2014)) and provided with a label-conditioned sampling mechanism (Odena et al. (2016), Mirza and Osindero (2014)) require tens of training epochs through the full data even for very large datasets and thus do not match the reactivity or learning efficiency requirements for online learning. Besides, the above-mentioned works only consider the case of *i.i.d.* data environments. In Sec. 3.5 we show that generative models also suffer from catastrophic forgetting when the *i.i.d.* assumption does not hold.

This issue is addressed in Zhai et al. (2019) by applying knowledge distillation (Hinton et al. (2015)) to perform lifelong learning in conditional GANs. This results in much lower forgetting than sequential fine-tuning. However, the proposed procedure only suits the incremental learning settings and cannot be generalized to learning from continuous unordered streams. Also, the method is validated on short sequences of up to five separate tasks, which does not demonstrate scalability to more complex problems.

A simple solution to obtain labeled data is to train an ensemble of generative models, one per class (Besedin et al. (2017)). However, such an approach has two major drawbacks. First, it goes against the scalability requirements and can face computational and memory issues for datasets with a large number of classes. Second, such an approach does not make use of the ability of deep models to share hidden representations across similar tasks, which has proven beneficial to faster and more effective learning (Girshick et al. (2016)).

In order to fit all the required characteristics, we propose to build the pseudo-rehearsal mechanism on the basis of autoencoders (Rumelhart et al. (1985)). In their classical formulation autoencoders consist of two parts: an encoder that maps the original data to a representation space of lower dimension

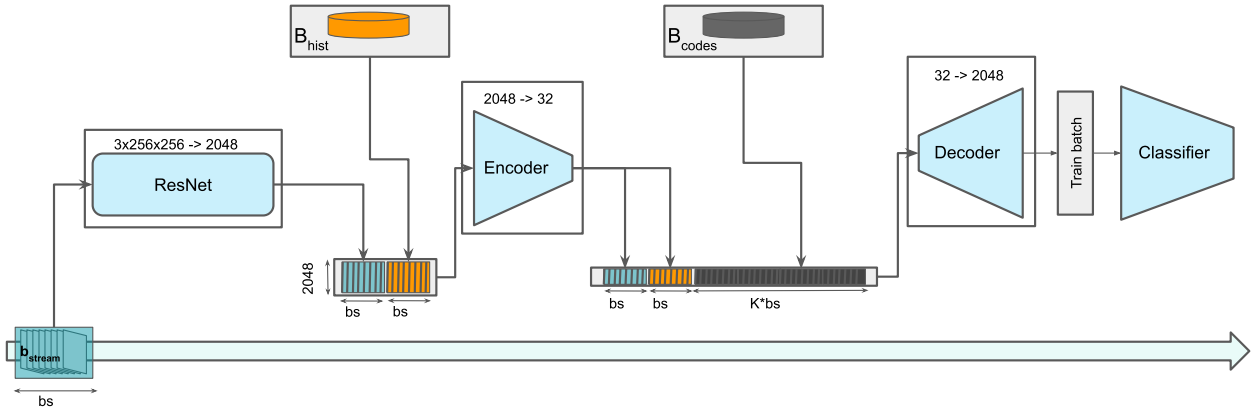


Fig. 1: Flowchart of the full proposed method. To avoid overload, only the forward pass is shown, without the updating of the memory buffers B_{hist} and B_{codes}. At a given time step, the system receives a batch of stream data b_{stream} and passes it through the ResNet feature extractor. Obtained feature vectors are then concatenated with additional samples from the historical data storage B_{hist} to prevent the distribution drifts in the auto-encoder and then passed through the encoder network. Resulting codes are mixed with the additional codes retrieved from the historical low-dimensional storage B_{codes} to ensure the presence of classes that are missing in the stream. The obtained batch of codes is then passed through the decoder and is used to train the classifier. Normally, stream data do not have to pass through the encoding-decoding process and can be directly fed to the classifier. However, this step is necessary in our case because the training procedure for the auto-encoder, described in Sec. 3.4 below, uses a loss function based on the output of the main classifier on the reconstructed data.

(the space of codes) and a decoder that reconstructs the original samples from these hidden representations. Standard autoencoders are not generative models since their output is not the result of sampling from a random distribution, but of recovering data samples from hidden representations. In Kingma and Welling (2013) the authors put forward variational autoencoders that aim at learning data distributions instead of memorizing samples and, as such, allow sampling in a purely generative way. To perform conditional sampling, the proposed model requires estimating the class-conditional distributions of latent codes, which can only be done offline and thus cannot be applied to an online learning scenario.

In our experiments we employ the encoding part of the model to produce codes, *i.e.* low-dimensional representations of the original data. The latest M encoded representations for each class are stored in a buffer B^{codes} for later use. The parameter M is determined by cross-validation and amounts to a small percentage of the test database in our experiments (see Sec. 4.3). Since the encoded representations are very low-dimensional, the storage requirements of this procedure are also very low. During stream training, we employ the decoder to reconstruct synthetic samples for missing data classes and join them to the stream data. We then train both the classifier and the autoencoder on the resulting batch of data, as discussed in Sec. 3.2. The full workflow is illustrated in Fig. 1. Compared to standard rehearsal-based approaches, autoencoders allow to significantly reduce working memory requirements because we only store the low-dimensional representations. In Sec. 4.2 we provide experimental results relating performance to the size of these autoencoder representations. At the same time, due to the dynamic nature of learning and continuous buffer updates, such a procedure prevents the classifier from overfitting on historical samples, which is one of the main issues of rehearsal mechanisms. Moreover, autoencoders provide a straightforward and accurate way to retrieve labeled data from a single model. In the next two subsections we propose enhancements to the autoencoder training procedure in order to improve its performance in this scenario.

3.4. Classification-based loss to train autoencoders

As mentioned at the beginning of this section, we want the reconstructed data produced by the autoencoder to influence the training process in the same way as the original data. To encourage this, we introduce an auxiliary loss that measures how far from each other are the outputs of the classifier (C) on the

original (x) and reconstructed data ($G(x)$):

$$\mathcal{L}_{cl} = \|C(x) - C(G(x))\|_2$$

We then train the autoencoders with the combined loss function

$$\mathcal{L} = \alpha_{cl}\mathcal{L}_{cl} + \alpha_{rec}\mathcal{L}_{rec} \quad (1)$$

where $\mathcal{L}_{rec}$ is the standard mean squared reconstruction error usually used to train the autoencoders and α_{cl}, α_{rec} control the trade-off between classification and reconstruction loss.

A similar additional classification loss to train Denoising Autoencoders was introduced in Zhou et al. (2012) with the main goal to perform a discriminative task when training the autoencoders on labeled data. Unlike what we propose, the authors plug the classification layer directly to the output of the encoder and train the encoding part of the network in a standard "classification task" way by fitting the one-hot label vectors. Such a procedure encourages the separation of classes in the code space rather than explicitly enforcing similar responses of the classifier on the original and reconstructed sample.

3.5. Dealing with distribution drifts in autoencoders

Since our system stores the most recent samples from the stream at each iteration as latent codes into the buffer B^{codes}, the system continuously updates the autoencoders with the newly arrived data. Over several iterations, this procedure tends to produce in the autoencoders a forgetting effect similar to the one of the main classifier: classes not present in the current set of updating samples tend to be overwritten by the present ones. To avoid this, when updating the autoencoders we need to include, for the missing classes, the data reconstructed from the codes already stored in the B^{codes} buffer.

However, recursively training the autoencoder on data it has reconstructed causes a drift of the distribution of the encoded samples, which slowly diverges from the distribution of real data. This process ends up producing samples that are harmful for training the classifier. Simply adding data from the short-term memory buffer does not solve the problem. To deal with this, we propose a penalization scheme that reduces the influence on training of the samples that have gone through many recursive reconstruction steps. For each code stored in the buffer we count the number of reconstructions r it has gone through (starting with 1 when the code is just produced from the stream data). For both the classifier and the autoencoder, when evaluating the loss for a given data batch, we compute the weighted average error using the reconstruction counters as weights in the following way:

$$L = \left(\sum_{i=1}^{k} \frac{1}{r_i}\right)^{-1} \cdot \left(\sum_{i=1}^{k} \frac{l_i}{r_i}\right)$$

where l_i and r_i are the loss and respectively the number of reconstructions for the sample i in the batch, and k the batch size. Older codes have their weights progressively decrease, which fits our purpose since they are the more likely to have drifted from the original distribution. If $r_i = 1$ for all i (*i.e.* all samples are equally important for training) we obtain the usual average loss over all the k samples in the buffer. Letting r_i infinitely grow, however, could result in an opposite effect where most of the stored codes have no influence on the training process. To avoid this, we set an upper bound $r_{max} = 10$ (value found by grid search) for the maximum number of re-generations.

4. Evaluation of online classification without forgetting

4.1. Experimental setup

In this section we present the experimental validation of our proposal. We start by describing the three datasets we use: the well-known MNIST dataset, broadly employed for prototyping computer vision algorithms, the LSUN dataset, a significantly larger and more complex benchmark, and Syn-1000, a synthetically-generated dataset which we use to evaluate the scalability of our proposal to a very large number of classes in the input stream.

We then evaluate our approach in a series of offline and online experiments. To show the ability of our method to alleviate catastrophic forgetting, we compare it to "pure stream" learning where training is performed, for each time interval, on the the data of that interval alone and no action is taken to avoid catastrophic forgetting. To measure the impact of the pseudo-rehearsal mechanism, we compare to standard rehearsal where a part of the training data is stored and reused. The results of our

continual learning framework are eventually compared to those of the reference method that consists in retraining the model, for each time interval of the stream, on the full set of historical data ("full rehearsal").

4.1.1. Datasets and data preparation

MNIST dataset. **MNIST** [3] is a collection of gray-scale images of hand-written digits, 28×28 pixels each. It consists of 10 classes, each corresponding to a separate digit from 0 to 9. The training and testing sets have 60000 and respectively 10000 images. In our experiments we additionally extract 10000 images from the training set to obtain a cross-validation set for hyper-parameter optimization. For each class, the training, validation and test sets have respectively 5000, 1000 and 1000 images each. MNIST dataset is used as the baseline proof-of-concept dataset in many Deep Learning papers.

Images in the original dataset are provided in byte numerical format. Prior to processing, all the pixels of every image in MNIST are set to the range $[-1, 1]$ by dividing them by $(255/2)$ and subtracting 1. This is a commonly used transformation that proved to significantly accelerate and stabilize training. Our goal here is not to challenge state-of-the-art offline classification results, but rather to show that our approach can learn from dynamic streams in a fast and efficient way. Thus, we decided to keep all the intermediate models and data representations as simple as possible. For this reason, we process MNIST images as unfold 1D vectors of 784 features each.

LSUN dataset and feature extraction. To test our method on larger and more complex data we employ the **LSUN** dataset with scenes[4] and objects[5]. This dataset consists of 30 classes (10 of scenes and 20 of objects) of natural RGB images with resolutions of 256×256 pixels or higher. Each class contains from 130K up to 3M images. We balance the classes in the training set by selecting 100K images per class (indices 1 to 100,000 in the database indexing system) for training. In the original data, a test set of 300 images per class is only provided for the 10 classes of scenes. As this is not sufficient to evaluate performance on such a large scale, we extract 10K images per class for the validation set (indices 100,001 to 110,000) and 10K for the test set (indices 110,001 to 120,000).

Testing our proposal directly on LSUN requires considerable computational resources. Our system does not use the images directly, but only the convolutional features obtained from a forward pass through a pre-trained network. To accelerate the process we first extract the convolutional features for all the images. We employ a ResNet-152 model[6] (He et al. (2016)), pre-trained on ImageNet, and store the 2048-dimensional features of the second to last hidden layer. In recent years, such a feature extraction procedure proved to efficiently transfer knowledge across different natural image datasets that share similar representations, and therefore produce class-representative features on new data.

Synthetic data generation. To see how our approach behaves when the number of classes increases, and thus when a large number of previously learnt classes are missing from the stream, we introduce a synthetic dataset, called **Syn-1000** in the following, with up to $N = 1000$ data classes. To keep things comparable with the previous experiments using natural images, we use the same number of features ($dim = 2048$).

In the proposed data synthesis procedure, each class follows a multi-dimensional Gaussian distribution. To generate the dataset we start by initializing the class centers μ_i for all the classes $i = \{1, \ldots, N\}$ in such a way that no two centers are closer than a threshold distance d: $\forall i \neq j, \left\|\mu_i - \mu_j\right\|_2 \geq d$. We then compute the covariance matrices Σ_i, which are positive semi-definite and thus can be represented as $\Sigma_i = O^T D_i O$, where O is an orthonormal matrix and D_i is diagonal with positive elements. To initialize the covariance matrices we first generate a random square matrix $M \in \mathcal{U}[-1, 1]^{dim \times dim}$ and perform Gram-Schmidt orthogonalization on it to obtain O. We then initialize random diagonal matrices D_i for all classes and

[3] http://yann.lecun.com/exdb/mnist/

[4] http://dl.yf.io/lsun/scenes/

[5] http://tigress-web.princeton.edu/ fy/lsun/public/release/

[6] https://pytorch.org/docs/stable/torchvision/models.html

Require: N : Number of classes in the dataset
Require: dim : Dimension of the data features
Require: d : Minimal distance between classes
  Initialize $\mu_1 \in \mathcal{U}[-1,1]^{dim}$
  for i in $range(2, N)$ **do**
    Initialize $\mu_i \in \mathcal{U}[-1,1]^{dim}$
    while $\min_{j<i} \left\|\mu_j - \mu_i\right\|_2 < d$ **do**
      for k in range$(1, i-1)$ **do**
        $\mu_i = \mu_k + d\frac{\mu_i - \mu_k}{\|\mu_i - \mu_k\|_2}$
      end for
    end while
  end for
  Initialize $M \in U[-1,1]^{dim \times dim}$
  O = Gram-Schmidt(M)
  for i in range(1, N) **do**
    $D_i = \text{diag}(\mathcal{U}(0.5,1]^{dim})$
    $\Sigma_i = O^T D_i O$
  end for
  return $\{(\mu_i, \Sigma_i)\}_{i=1}^N$

Algorithm 2: Syn-1000 dataset initialization. The notation $x \in \mathcal{U}(\mathcal{A})$ means that the value x is sampled from the uniform distribution on the set $\mathcal{A}$.

transform them with the help of O to obtain Σ_i. Therefore we can now sample data for N classes from $\mathcal{N}_{dim}(\mu_i, \Sigma_i)$, see Alg. 2.

4.1.2. Model architectures and optimization setup

All the models in this study are implemented with PyTorch and trained on NVIDIA 1080 Ti GPUs. Due to the constraints mentioned in Sec. 3.3, in our experiments we use shallow models that are easy to implement and fast to train. Their architectures are provided in Table 1, with **FC** standing for fully-connected linear layers, **ReLU** for Rectified Linear Unit (Nair and Hinton (2010)), **bn** for 1D batch normalization (Ioffe and Szegedy (2015)) and **cs** for code size in autoencoders. For each dataset, both the autoencoder and the classifier are separately optimized using Adam (Kingma and Ba (2014)) with $\alpha = 10^{-3}$, $(\beta_1, \beta_2) = (0.9, 0.999)$ and weight decay $= 10^{-5}$. For the LSUN dataset, the input is the 2048 dimensional feature vector provided by ResNet-152 as described in Sec. 4.1.1. When training is performed in a static or offline scenario, the results obtained on the MNIST and LSUN datasets are close to the state of the art as shown in the next subsection.

4.2. Offline experiments

Our online learning framework (see Sec. 3.2) depends on a number of hyper-parameters such as s^{hist} (size of the historical buffer that stores most recent samples), s^{codes} (size of the buffer that stores auto-encoded samples), cs (size of the hidden codes layer in the autoencoder) and $(\alpha_{cl}, \alpha_{rec})$ (the trade-off between classification and reconstruction loss for autoencoder training). The static parameters cs and $(\alpha_{cl}, \alpha_{rec})$ are optimized by cross-validation in the offline scenario, while the sizes of the buffers are optimized in an incremental learning scenario that matches the required characteristics of the online learning system.

For each dataset we select hyper-parameters that provide the best classification scores on corresponding validation subsets, as described in the following subsections.

4.2.1. Impact of the classification loss term

We first adjust the tradeoff between classification and reconstruction loss in the objective function used to train the autoencoder, Eq. (3.4), by performing a grid search on $(\alpha_{cl}, \alpha_{rec}) \in (0, 10)^2$. The grid search is conducted for each dataset by training autoencoders with a code size of 32 and cross-validating each pair $(\alpha_{cl}, \alpha_{rec})$. One of the most important criteria for the choice of the generative model in our online learning approach is the training reactivity, *i.e.* the ability of the model to learn fast and perform well when trained with few data. Therefore, we select the values of $(\alpha_{cl}, \alpha_{rec})$ for which the model trained for a single epoch on the training set gives the best average accuracy on the validation set over 10 independent runs. We obtained the following trade-off values for $(\alpha_{cl}, \alpha_{rec})$: $(0.001, 0.003)$ for MNIST, $(0.1, 1)$ for LSUN and $(0.01, 3)$ for Syn-1000.

Syn-1000 and MNIST being quite simple, the use of the reconstruction loss alone is almost enough to achieve optimal performance. For this reason, full cross-validation performance for

Data	Classifier	Autoencoder		Params
		Encoder	Decoder	
MNIST	FC(784,256)→ReLU→ FC(256,64)→ReLU→ FC(64,10)	FC(784,256)→bn→ReLU→ FC(256,cs)→bn→ReLU	FC(cs,256)→bn→ReLU→ FC(256,784)	0.65M
LSUN, Syn-1000	FC(2048,784)→ReLU→ FC(784,256)→ReLU→ FC(256,30)	FC(2048,512)→bn→ReLU→ FC(512,128)→bn→ReLU→ FC(128,cs)	FC(cs,128)→bn→ReLU→ FC(128,512)→bn→ReLU→ FC(512,2048)	4M

Table 1: The architectures of the classifier and autoencoder used in this study for the MNIST, LSUN and Syn-1000 datasets.

$\alpha_{cl} \setminus \alpha_{rec}$	0	0.001	0.003	0.01	0.03	0.1	0.3	1	3	10
0	x	3.4	3.6	5.7	10.7	17.3	21.6	25.2	26.0	27.1
0.001	75.1	77.3	79.2	78.8	77.7	76.9	67.5	60.2	52.5	42.6
0.003	82.0	81.2	82.1	82.3	82.0	81.0	78.7	72.3	63.4	52.7
0.01	83.8	84.1	83.0	82.4	83.1	83.3	82.8	80.1	74.6	64.4
0.03	83.1	84.7	84.2	83.9	83.2	83.9	84.0	83.4	80.8	73.0
0.1	85.1	83.5	84.8	84.5	84.8	84.7	83.6	**85.3**	83.8	82.3
0.3	84.9	84.3	84.5	84.8	84.7	83.5	84.1	85.0	83.2	84.4
1	84.5	83.8	84.3	84.2	84.5	83.8	84.6	85.0	84.5	83.9
3	84.2	83.6	85.0	84.1	84.7	84.1	84.1	84.4	84.8	84.6
10	84.1	84.6	84.5	82.6	84.6	84.5	85.1	84.9	84.2	84.8

Table 2: Validation set accuracy on the pretrain part of the LSUN dataset, obtained during the grid search aimed to optimize the trade-off between classification and reconstruction losses in the autoencoders.

Code size	2	4	8	16	32	Baseline
MNIST	91.3	95.3	97.1	**97.7**	98.2	99.4
LSUN	57.9	72.9	79.5	83.3	**89.1**	90.7
Syn-1000	79.8	91.1	93.3	**94.5**	94.6	96.1

Table 3: Accuracies obtained on the validation sets of MNIST, LSUN and Syn-1000 depending on the code size employed by the autoencoders. The baseline dimension is 784 for MNIST, and 2048 for LSUN and Syn-1000.

each pair $(\alpha_{cl}, \alpha_{rec})$ is shown in Table 2 only for LSUN. As we see from the table, the classification term of the loss is very useful for more difficult datasets like LSUN. For the latter, the gradient induced by the classification term of the loss dominates the reconstruction one after a few epochs of training and, as such, contributes to creating more representative codes, increasing performance.

These values allow us to make an interesting observation about how the influence of each term in the loss function (3.4) changes depending on the nature of the data. While for the MNIST and Syn-1000 datasets the reconstruction error alone is enough to obtain high accuracy (we only get small improvements by adding the classification term), on LSUN the feature reconstruction criterion alone results in an accuracy of only 27%, compared to 85.5% with the optimal combination of two losses. We believe that this effect is due to the extremely small norms of the feature vectors extracted by ResNet from LSUN, which results in small L_2 distances between the data samples and the corresponding reconstructions, leading to van-

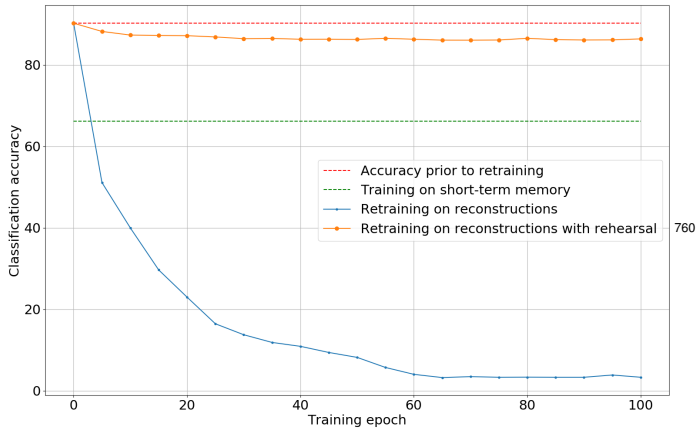


Fig. 2: On the LSUN dataset, the effect of growing reconstruction error in autoencoders (leading to catastrophic forgetting) when trained on their own reconstructions (blue line) and the positive effect rehearsal has on this process (orange line).

ishing gradients. Classification loss, in turn, computes the error in the space of classifier output. This involves a supplementary mapping that ends up by solving the problem of small gradients. Therefore, the classification term in the autoencoder objective function, in addition to its main use, also performs an adaptive parameter normalization in the hidden layers of the model.

4.2.2. *Code size in the autoencoders*

As described in Sec. 3.3, we employ autoencoders to reduce the overall storage requirements of the learning system. In the proposed online learning scenario, one has to store the hidden encoded representations for the duration of the stream. Therefore, the choice of the parameter cs (code size) has a direct influence on the storage requirements. For each dataset we perform a grid search over cs values in an offline manner. We train the classifier and the autoencoder on the training split, and evaluate them on the validation split. The goal of this search is to find the smallest value of cs that provides an accuracy higher than a predefined threshold that we set to 98% of the offline classification performance. The results of this evaluation, averaged over 10 independent runs, are shown in Table 3 with the selected values highlighted in bold.

4.2.3. *Impact of short-term memory on autoencoder behavior*

As discussed in Sec. 3.2 and 3.5, using pseudo-rehearsal alone to train models in an online setting can result in progressively growing error in the multi-class generator. While adding some real historical data to the pseudo-rehearsal data can solve the problem, storing and reusing historical data for large-scale online learning can be heavy and computationally inefficient. In order to limit the storage requirements of the online learning system we bound the maximum size of the historical buffer B^{hist} to 1% of full data size. A natural question arises: the stored historical data alone are not enough to train all the models without using more sophisticated methods? In this section we investigate how autoencoders act when trained on their own reconstructions and show that the short-term memory mechanism (using B^{hist}), while not sufficient to solve the problem, does help the autoencoder to avoid forgetting in the online scenario.

To evaluate the impact of short-term memory, we first pretrain a classifier and a multi-class autoencoder until convergence. We then pass the full training set through the autoencoder, replace the data by their reconstructions and retrain the autoencoder on them. This process is repeated for 100 epochs. To measure the forgetting effect, we compute the accuracy of the classifier on the reconstructed validation set after each training epoch. To evaluate the importance of historical data for the described process, we redo the same experiment preserving a small portion of original data for each class (B^{hist} buffer). Finally, to demonstrate that short-term memory alone is not enough to reach desired performance, we train the autoencoder from scratch on B^{hist} and register the maximum classifier accuracy on the reconstructed validation set. Since for the MNIST dataset a randomly initialized classifier trained offline on no more than 50 images per class (size of the stream batch in our online experiments) provides an accuracy of more than 85%, we confirmed this hypothesis on the LSUN dataset.

Fig. 2 shows the results of this evaluation. We can clearly see that retraining the autoencoder on its own reconstructions (blue line) results in a fast decrease of classification accuracy, while the addition of a small portion of the original data to the

reconstructed data (orange line) stabilizes training and allows to avoid forgetting in the autoencoder. Training models separately either on the reconstructed data or on a small portion of the original data results in forgetting in the first case or overfitting in the second. Using them together solves both problems while requiring only a minimal computation overhead.

4.3. Continual learning from Dynamic Data Streams

In this section we present the validation of our full proposal on the MNIST, LSUN and Syn-1000 datasets simulating non-i.i.d. streams (see Sec. 3.1). A stream is divided in time intervals I_t, each formed of 100 (for MNIST) or 300 (for LSUN and Syn-1000) sequentially arriving batches, of size $bs = 50$ samples each, from at most 10% of all the available classes (1, 3 and 100 classes for MNIST, LSUN and respectively Syn-1000).

As described in Sec. 3.1, part of the incoming data is stored for later use: the buffer B^{hist} keeps the latest 50 raw samples from each class, while B^{codes} keeps the latest 5,000 encoded representations from each class. These values are obtained by cross-validation, as explained in Sec. 4.2. For each batch from the stream we randomly sample a single batch from B^{hist} and $K = 18$ batches reconstructed from B^{codes} by the autoencoder, concatenate them and feed the resulting batch of size 1000 ($= (1 + 1 + 18) \times 50$) to the learning system (see Alg. 1).

For each dataset we simulate the stream for 1000 intervals and compare the performance of 5 methods: naive training on stream data alone (*Pure stream*), training with stream data and limited rehearsal *i.e.* injections of data only from B^{hist} (*Rehearsal*) and three configurations of the proposed approach — the autoencoder G trained using the reconstruction loss only (*Our method,* $\alpha_{cl} = 0$), using the classification loss only (*Our method,* $\alpha_{rec} = 0$) and using the mixture of classification and reconstruction losses (*Our method,* α_{cl}, α_{rec}). At the end of each interval I_t we measure the classification accuracy obtained by the current classifier on the test set. Fig. 3 and Fig. 4 show the results of these continuous learning experiments.

With no surprise, naive online training on stream data results in unrecoverable catastrophic forgetting for each dataset. An interesting observation can be made on the Syn-1000 dataset, Fig. 3(b). After 150 stream intervals the overall performance starts improving and saturates after 500 intervals at an accuracy of about 70%. This phenomenon can be explained by the nature of the synthetic classes, that are rather well separated high dimensional Gaussians. Performance is low in the beginning when many new classes are added, but after the majority of classes have been seen the model is gradually converging towards a sub-optimal solution.

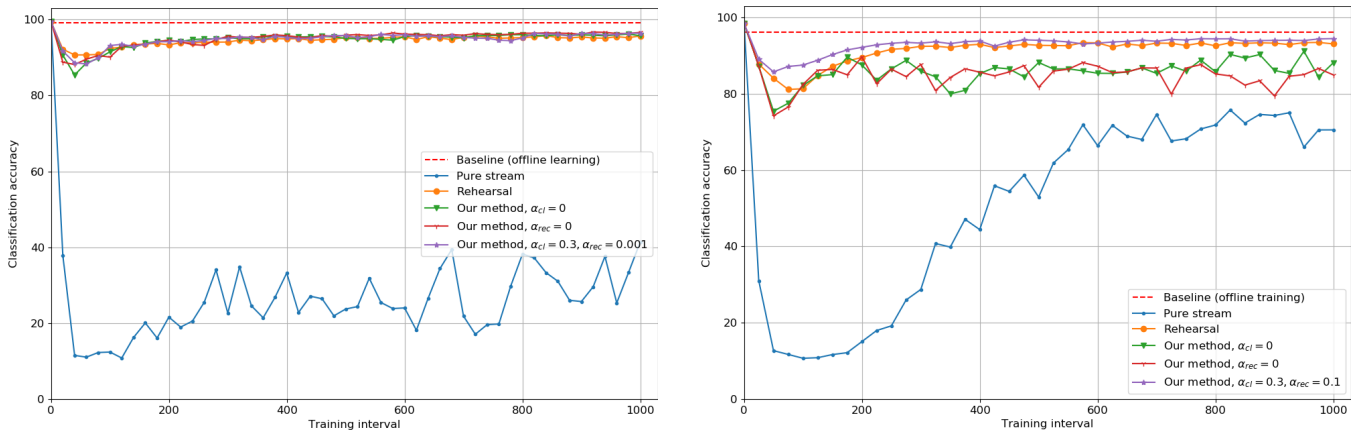
For similar reasons, limited rehearsal performs well on Syn-1000, resulting in stable and accurate learning comparable to our main method and approaching the offline learning baseline (relative error within 3%). The experiments on Syn-1000 nevertheless show that our approach efficiently scales to problems with a large number of separate classes.

On the MNIST dataset one can see in Fig. 3(a) that our method and limited rehearsal have similar performance (accuracy of about 96.5%), that is $\leq 4\%$ lower than the offline learning baseline. As for the Syn-1000 dataset, this is due to the fact that the MNIST dataset is easy for the employed classifier. However, the point of this experiment is that the catastrophic forgetting phenomenon is very strong (blue line for pure stream training) even for such a simple database.

Among the three datasets, LSUN is the most difficult in terms of class complexity. The results shown in Fig. 4 confirm that the full approach proposed in this paper, with the mixed loss function to train the autoencoder and the mechanism to compensate the negative impact of recursive reconstructions, significantly outperforms limited rehearsal and the other approaches based on a single-term loss function. On the test set, for the last 100 stream intervals, our approach reaches an average accuracy that is only 5.7% lower than the offline training baseline. To the best of our knowledge, our results can be considered as state-of-the-art in online classification training on the LSUN dataset simulated as a continuous non-i.i.d. stream.

4.4. Complexity analysis

Table 4 provides the time and memory cost comparison between our method and the evaluation baselines. In contrast to full rehearsal, our method has constant training time per stream



(a) MNIST (b) Syn-1000

Fig. 3: Stream training on MNIST (left) and Syn-1000 (right). The results are shown for naive learning from stream data alone (blue), learning with limited rehearsal (yellow), our method with the autoencoder trained with reconstruction loss only (green), with classification loss only (red) and full method (purple).

Method	Time cost	Storage cost
Pure stream	n	–
Full rehearsal	$t \cdot n$	$ds \cdot t \cdot n$
Rehearsal	$\lvert B^{hist}\rvert + n$	$ds \cdot \lvert B^{hist}\rvert$
Our method	$(K+2) \cdot n$	$cs \cdot \lvert B^{codes}\rvert + ds \cdot \lvert B^{hist}\rvert$

Table 4: Cost comparison between naive learning from stream data alone, learning with full rehearsal (*i.e.* with all past data in each time interval), with limited rehearsal, and our full method. Cost is evaluated for time interval I_t, t is the number of past time intervals, n is the average amount of stream data per time interval, ds is the size of a data item, cs is the size of autoencoder codes, $|B|$ is the size of a buffer B and K is the generated-to-real data ratio.

data batch, which makes it scalable to large multi-class problems. Also, it has only a constant overhead over the pure stream training time. The storage cost of our method is significantly lower than for full rehearsal due to the bounded size of the buffers and the low dimension of the encoded features we store.

5. Conclusion and perspectives

In this paper we put forward a more precise formulation of the problem of online learning from unordered non-i.i.d. data streams. We discussed the requirements one has to impose to the learning system in order to solve such a problem and then proposed a model that has all the desired characteristics. Our approach alleviates catastrophic forgetting in online learning settings by making use of pseudo-generative models implemented as autoencoders. When necessary, we approximate missing real data from the stream by using stored low-dimensional codes. This is a good compromise allowing us to avoid building online generative models that can be dynamically updated (a difficult task for high-dimensional complex distributions).

We also argued that instead of aiming for a precise reconstruction of real data we can train autoencoders to extract data representations that make sense for an application of interest (classification in our case). We introduced an objective function that helps pseudo-generative models capture classification-relevant information.

Our approach shows stable learning (low variation in validation results) and state-of-the-art performance on the unordered non-i.i.d. streams obtained from the MNIST and LSUN datasets. We further demonstrated the scalability of the proposed approach to the case of a much larger dataset with 1000 distinct classes.

For large-scale real-life applications, an online learning system must be able to perform efficiently in a changing environment and should be capable of adapting its architecture to a dynamically expanding environment. Moreover, such a model should preserve acquired knowledge not only by approximating and replaying historical data, but also by imposing rele-

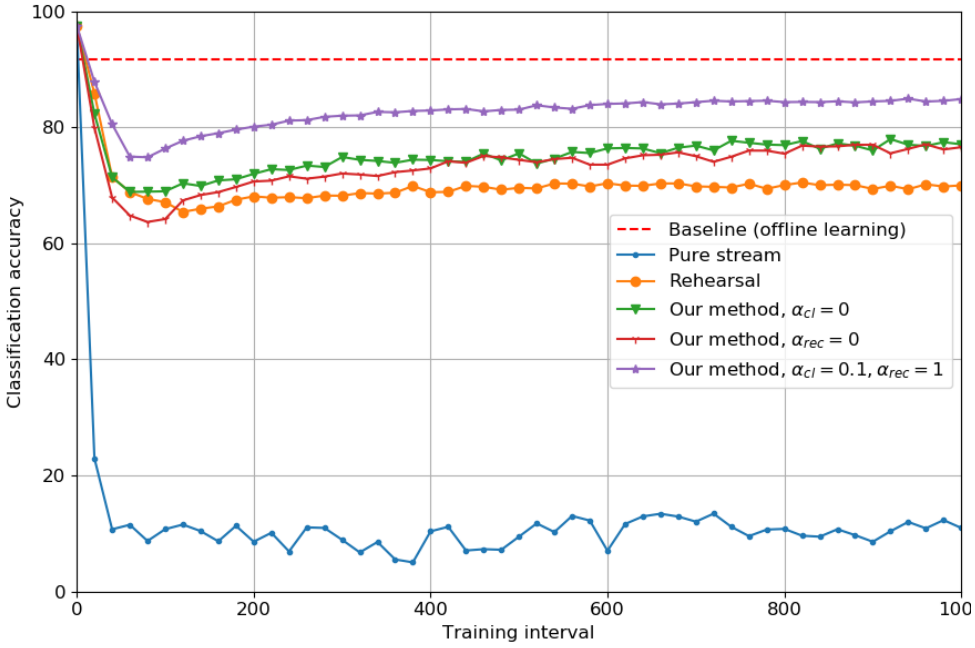


Fig. 4: Stream training on LSUN. The results are shown for naive learning from stream data alone (blue), learning with limited rehearsal (yellow), our method with the autoencoder trained with reconstruction loss only (green), with classification loss only (red) and full method (purple).

vant constraints on its parameter space. We, therefore, believe that the potentially best approach to perform human-like lifelong learning should combine all the three, for now separate, types of online learning approaches: evolving architectures, regularization-based and dual-memory-based.

References

Arjovsky, M., Chintala, S., Bottou, L., 2017. Wasserstein GAN. arXiv preprint arXiv:1701.07875 .

Barratt, S., Sharma, R., 2018. A note on the inception score. arXiv preprint arXiv:1801.01973 .

Besedin, A., Blanchart, P., Crucianu, M., Ferecatu, M., 2017. Evolutive deep models for online learning on data streams with no storage, in: ECML/PKDD Workshop on Large-scale Learning from Data Streams in Evolving Environments (IOTSTREAMING-2017).

Besedin, A., Blanchart, P., Crucianu, M., Ferecatu, M., 2018. Deep online storage-free learning on unordered image streams, in: ECML/PKDD Workshop on Large-scale Learning from Data Streams in Evolving Environments (IOTSTREAMING-2018).

Calandra, R., Raiko, T., Deisenroth, M., Pouzols, F., 2012. Learning deep belief networks from non-stationary streams. Artificial Neural Networks and Machine Learning–ICANN 2012 , 379–386.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P., 2011. Natural language processing (almost) from scratch. Journal of Machine Learning Research 12, 2493–2537.

Domingos, P., Hulten, G., 2000. Mining high-speed data streams, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 71–80.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D., 2017. Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint arXiv:1701.08734 .

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2016. Region-based convolutional networks for accurate object detection and segmentation. IEEE transactions on pattern analysis and machine intelligence 38, 142–158.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in Neural Information Processing Systems, pp. 2672–2680.

He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask R-CNN. arXiv preprint arXiv:1703.06870 .

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.

Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 .

Hou, S., Pan, X., Loy, C.C., Wang, Z., Lin, D., 2019. Learning a unified classifier incrementally via rebalancing, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Ioffe, S., Szegedy, C., 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 .

Kamra, N., Gupta, U., Liu, Y., 2017. Deep generative dual memory network for continual learning. arXiv preprint arXiv:1710.10368 .

Kemker, R., Kanan, C., 2017. Fearnet: Brain-inspired model for incremental learning. arXiv preprint arXiv:1711.10563 .

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Kingma, D.P., Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 .

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al., 2017. Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences , 201611835.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

Li, Z., Hoiem, D., 2017. Learning without forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence .

McCloskey, M., Cohen, N.J., 1989. Catastrophic interference in connectionist networks: the sequential learning problem. Psychology of learning and motivation 24, 109–165.

Mirza, M., Osindero, S., 2014. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 .

Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y., 2018. Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957 .

Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th international conference on machine learning (ICML-10), pp. 807–814.

Nguyen, H.L., Woon, Y.K., Ng, W.K., 2015. A survey on data stream clustering and classification. Knowledge and information systems 45, 535–569.

Odena, A., Olah, C., Shlens, J., 2016. Conditional image synthesis with auxiliary classifier GANs. arXiv preprint arXiv:1610.09585 .

Oza, N.C., 2005. Online bagging and boosting .

Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S., 2018. Continual lifelong learning with neural networks: A review. arXiv preprint arXiv:1802.07569 .

Rai, P., Daumé III, H., Venkatasubramanian, S., 2009. Streamed learning: One-pass svms., in: IJCAI, pp. 1211–1216.

Robins, A., 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science 7, 123–146.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1985. Learning internal representations by error propagation. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R., 2016. Progressive neural networks. arXiv preprint arXiv:1606.04671 .

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X., 2016. Improved techniques for training GANs, in: Advances in Neural Information Processing Systems, pp. 2234–2242.

Seidl, T., Assent, I., Kranen, P., Krieger, R., Herrmann, J., 2009. Indexing density models for incremental learning and anytime classification on data streams, in: Proceedings of the 12th international conference on extending database technology: advances in database technology, ACM. pp. 311–322.

Sermanet, P., Kavukcuoglu, K., Chintala, S., LeCun, Y., 2013. Pedestrian detection with unsupervised multi-stage feature learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3626–3633.

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

Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks, in: Advances in neural information processing systems, pp. 3104–3112.

Webb, G.I., Hyde, R., Cao, H., Nguyen, H.L., Petitjean, F., 2016. Characterizing concept drift. Data Mining and Knowledge Discovery 30, 964–994.

Yoon, J., Yang, E., Lee, J., Hwang, S.J., 2018. Lifelong learning with dynamically expandable networks .

Yu, J., Tan, M., Zhang, H., Tao, D., Rui, Y., 2019. Hierarchical deep click feature prediction for fine-grained image recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence , 1–1doi:10.1109/TPAMI.2019.2932058.

Yu, J., Zhang, B., Kuang, Z., Lin, D., Fan, J., 2017. iPrivacy: Image privacy protection by identifying sensitive objects via deep multi-task learning. IEEE Transactions on Information Forensics and Security 12. URL: http://par.nsf.gov/biblio/10026310, doi:10.1109/TIFS.2016.2636090.

Zenke, F., Poole, B., Ganguli, S., 2017. Continual learning through synaptic intelligence. arXiv preprint arXiv:1703.04200 .

Zhai, M., Chen, L., Tung, F., He, J., Nawhal, M., Mori, G., 2019. Lifelong GAN: Continual learning for conditional image generation, in: International Conference on Computer Vision (ICCV).

Zhang, J., Zhang, J., Ghosh, S., Li, D., Tasci, S., Heck, L.P., Zhang, H., Kuo, C.J., 2019. Class-incremental learning via deep model consolidation. CoRR abs/1903.07864. URL: http://arxiv.org/abs/1903.07864, arXiv:1903.07864.

Zhou, G., Sohn, K., Lee, H., 2012. Online incremental feature learning with denoising autoencoders. Ann Arbor 1001, 48109.