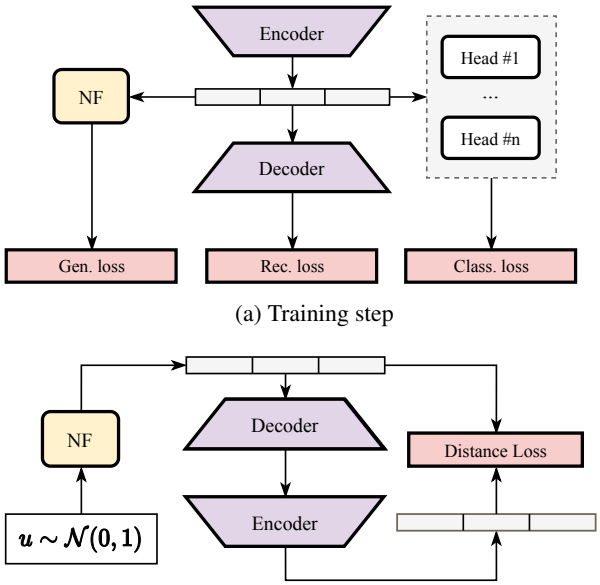
Pseudo-Rehearsal for Continual Learning with Normalizing Flows

Jary Pomponi [*1] **Simone Scardapane** [*1] **Aurelio Uncini** [1]

Abstract

Catastrophic forgetting (CF) happens whenever a neural network overwrites past knowledge while being trained on new tasks. Common techniques to handle CF include regularization of the weights (using, e.g., their importance on past tasks), and rehearsal strategies, where the network is constantly re-trained on past data. Generative models have also been applied for the latter, in order to have endless sources of data. In this paper, we propose a novel method that combines the strengths of regularization and generative-based rehearsal approaches. Our generative model consists of a normalizing flow (NF), a probabilistic and invertible neural network, trained on the internal embeddings of the network. By keeping a single NF conditioned on the task, we show that our memory overhead remains constant. In addition, exploiting the invertibility of the NF, we propose a simple approach to regularize the network's embeddings with respect to past tasks. We show that our method performs favorably with respect to state-of-the-art approaches in the literature, with bounded computational power and memory overheads.



(a) Training step

(b) Regularization step

Figure 1. Proposed algorithm for continual learning. (a) Training step, in which both the classifier network (encoder with the current head) and the generative model (NF with the decoder) are trained: the first one to solve the task, and the second one to replicate the samples from all the past tasks. (b) Regularization step, in which the encoder is regularized so that the new extracted embeddings are as close as possible to the past ones.

1. Introduction

One of the major open problems in deep learning is the so-called catastrophic forgetting (CF) (McCloskey & Cohen, 1989; Roger, 1990; French, 1999). It is the tendency of a NN to forget past learned information when training on new tasks. This problem is intrinsically connected with the continual learning (CL) property of a NN, which is the ability of a NN to learn consecutive information without forgetting previously stored knowledge.

Overcoming, or mitigating, CF is a key step in order to achieve a more general artificial intelligence; a system should be able to learn a sequence of tasks and remember them, following the lifelong learning paradigm (Thrun & Mitchell, 1995). This is a key problem because real world tasks continually evolve and, often, it is not possible to train a NN from scratch. Without efficient methods to overcome CF, online training in a lifelong learning scenario is not possible. Recently, there has been a resurgence of interest in this area due to its importance.

One of the first attempts to mitigate CF consisted in storing past examples and replaying them into the model while learning new information (Robins, 1995). These *rehearsal* methods have been improved over the years, with more complex memory systems and hybrid approaches emerging. In particular, many *pseudo-rehearsal* methods have been

[*]Equal contribution [1]Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Italy. Correspondence to: Jary Pomponi <jary.pomponi@uniroma1.it>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020. Copyright 2020 by the author(s).

proposed, in which the external memory is replaced with a generative model, capable of generating endless samples from the past. Pseudo-rehearsal algorithms, however, require generative models that are both easy to train (in order to provide high-quality data) and simple to condition on the task (in order to avoid having one generative model for each task). Both of these conditions are extremely challenging in practice.

To overcome (or complement) these limitations, many *regularization* methods have been studied over the last years, in which additional loss terms are used to mitigate CF. In elastic weight consolidation (EWC, (Kirkpatrick et al., 2017)), for example, past weights are used to regularize the training process, by slowing down the modification of important weights, where the importance is quantifed based on a Fisher information criterion. Alternative regularization strategies are achieved by acting on the previous outputs (Li & Hoiem, 2017) or gradients (Lopez-Paz & Ranzato, 2017). Recently, it was also shown that state-of-the-art performance can be achieved by acting on the internal embeddings (i.e., the activations before the classification layer) (Pomponi et al., 2020). A more complete overview of CL methods is provided later on in Section 2, or in (Parisi et al., 2019).

In this paper, we propose a novel CL method aiming to combine the benefits of pseudo-rehearsal and regularization strategies. Our algorithm is shown in Fig. 1a, and can be summarized in three steps:

1. Similarly to pseudo-rehearsal strategies, we store information on past tasks by training an auxiliary generative model. Instead of training it on the input space, however, we train it to generate samples from the internal embeddings of the network (the output of the last convolution layer), simultaneously with the main classifier.

2. We use a normalizing flow (NF, (Papamakarios et al., 2019)) as generative model. NFs are invertible neural networks that can perform sampling and density estimation in both directions. In this way, the NF can be trained efficiently, with no need for additional components such as in generative adversarial networks.

3. Finally, we use the sampled embeddings from the trained NF to perform regularization with respect to the past tasks, as shown in Fig. 1b.

We claim that (i) training the generative model in the embeddings' space is significantly easier (both in its design and in its optimization) compared to the input space, and (ii) that regularization makes better use of past information stored in the generative model, compared to simply augmenting the mini-batch with new data (similarly to (Pomponi et al., 2020)). In our experimental evaluation, we validate these two claims, and we show that our method performs favorably (or better) than several state-of-the-art approaches, while requiring significantly less memory and computational overhead.

2. Related Works

The methods for overcoming CF can be categorized, in line with (Parisi et al., 2019) and (Maltoni & Lomonaco, 2019), in three broad groups. We underline that the boundaries are not always defined, with many methods, including ours, exploiting two or more of these strategies.

- **Architectural Strategies**: methods that use specific architectures, layers, activation functions and/or weights freezing/pruning, and eventually grow the architecture when needed (e.g., (Rusu et al., 2016)). For example, Hard Attention on Task (HAT, (Serra et al., 2018)) uses an attention mechanism in order to route the information/gradient flow in the NN and preserve the weights associated to past tasks.

- **Rehearsal strategies**: in this case, past examples are stored and later replayed in the current mini-batches to consolidate the network. In order to avoid having to explicitly store past examples, which requires a growing memory, *pseudo-rehearsal* algorithms (Robins, 1995) craft them on-the-fly, most notably exploiting generative models (Shin et al., 2017).

- **Regularization techniques**: in this case, popularized by elastic weight consolidation (EWC, (Kirkpatrick et al., 2017)), the loss function on the current task is extended with a regularization penalty to selectively consolidate past information or slow the training on new tasks. Broadly speaking, regularization methods are easy to implement, but they require to carefully select what information is being regularized, and how.

The method we propose in this paper is at the boundary of pseudo-rehearsal and regularization strategies, so we focus on these two classes below. In fact, our method use a generative model in the embedding space and then use the generated embeddings to regularize the network while training on new tasks.

Learning Without Forgetting (LWF, (Li & Hoiem, 2017)) is one of the first regularization methods. It attempts to alleviate CF by stabilizing the output layer using knowledge distillation. Others well-known regularization methods are EWC, which applies a soft structural regularization computed between the weights' importance relative to the past tasks and the current weights, and Synaptic Intelligence (SI, (Zenke et al., 2017)), a modification of EWC, which uses the difference between the current weights and their trajectory calculated during the training. Other methods

include Gradient Episodic Memory (GEM, (Lopez-Paz & Ranzato, 2017)), Averaged-GEM (Chaudhry et al., 2019), and the recently proposed Embedding Regularization (ER, (Pomponi et al., 2020)).

In GEM, the external memory is populated with past examples that are used to regularize the direction of the current gradients, in order to move the weights in a region of the space in which all the tasks are satisfied. This method is capable of improving past scores, but it requires to solve a complex minimization problem at every step, which does not scale well with the number of tasks. ER is a regularization technique in which the external memory contains past examples and their associated embeddings, extracted at the end of the training process on the associated task. The memory is used to impose a penalty to constrain the current embeddings to lie in the vicinity of the past ones. This method is extremely fast and requires little memory. The approach we propose in this paper follows the philosophy of ER to act at the level of the embeddings, instead of single weights or outputs. Another novel and interesting approach is proposed in (Ebrahimi et al., 2020), in which the authors use an Adversarial Continual Learning (ACL) approach: it aims to alleviate CF by learning a disjoint latent space representation composed of a task-specific latent space for each task and a task-invariant feature space for all tasks.

A more challenging set of methods are pseudo-rehearsal ones. In (Shin et al., 2017), the authors proposed a method which consists of two modules: a deep generative model and a task solver. In this way, samples from past tasks can be generated using the generative model and interleaved with information from the new tasks; the solver is used to predict the label associated to the generated images and then regularize the network. In (Kang et al., 2020) a similar approach, but based on a Variational Autoencoder (VAE), is proposed: it consists in a VAE and an external NN, which learns to replicate the distribution of the embeddings associated to a task; this external NN can be used to generate images associated to past tasks and reduce CF. Right now, pseudo-rehearsal methods were strictly evaluated on datasets of relatively low complexity. The question on whether these generative approaches can scale up to more complex domains is still open.

The method we propose here can be considered a pseudo-rehearsal one, but we focus on a more recent class of generative models (Kingma & Dhariwal, 2018), and we apply them at the level of embeddings instead of in the input space. We note that in the literature on generative models, a number of authors have considered similar combinations of autoencoders with a generative model on their latent space. In (Rezende & Mohamed, 2015), the authors applied a NF to learn a VAE prior. This idea has been further studied in (Kingma et al., 2016), where the authors proposed a new NF which scales well for high-dimensional embedding spaces. More similar to our proposal, in (Guo et al., 2019) the authors proposed a model which uses an adversarial generative model in the embedding space to generate high resolution images.

A relatively new and emerging area of study researches how to mitigate CF when the tasks' boundaries are not known. In this area we highlight (Aljundi et al., 2019), in which the authors proposed a task free approach to continual learning, using a regularization-based memory. In (Zeno et al., 2018), a task agnostic Bayesian method was proposed, demonstrating the ability of probabilistic models to handle ambiguous task boundaries. Finally, (Rao et al., 2019) has introduced the novel idea of unsupervised learning in a lifelong scenario.

Many other methods exists; for a complete review of existing methods see (Parisi et al., 2019).

3. Proposed method

3.1. Motivation

In a sense, rehearsal methods are close to optimal, because in the limit of a very large memory they recover a standard multi-task setting. On the other hand, these methods require a memory that, usually, grows linearly with the number of tasks. Additionally, the computational requirements increase with the number of tasks, since each past task needs to be remembered (and rehearsed) separately.

Pseudo-rehearsal methods try to overcome these limitations by substituting the memory with a generative model and doing parameter sharing on the generative model, which is incrementally trained on all the tasks and constrained to remember the information about the tasks encountered so far. However, this creates a new set of challenges: (i) the CF problem is removed from the NN, but the generative model itself potentially suffers CF; (ii) doing parameter sharing on real-world images can be difficult.

The aim of this paper is to propose a generative approach which does not work directly on the input, and that can be used to regularize the model instead of simply augmenting the dataset. The key idea of the proposed pseudo-rehearsal embedding regularization (PRER) is to use a generative model to sample new embeddings associated to past tasks. To regularize the network, we (i) reconstruct the images associated to the embeddings, (ii) calculate the embeddings given by the current network; (iii) force the old and the new embeddings to be as close as possible (by moving the current one in the direction of the past ones).

3.2. Problem formulation

The following formulation of the CL scenario is similar to the one proposed in (Lopez-Paz & Ranzato, 2017). We receive a sequence of tasks $t = 1, \dots, M$, each one composed by a set of triples $\{(\mathbf{x}_i, t, y_i)\}_{i=1}^{S} \in \mathcal{X} \times \mathbb{N}^+ \times \mathcal{Y}$, where $\mathbf{x}_i$ is a sample, t is an integer identifying the current task, and y_i is the corresponding label of $\mathbf{x}_i$. Within this formulation, the tasks never intersect, and a new task is collected only when the current one is over. The input and the labels can belong to any domain, although most benchmarks in the CL literature have considered the image domain in a classification setting (Parisi et al., 2019).

In this paper we focus on a CL scenario in which each task has its own classifier, meaning that only a portion of the NN, that we call the encoder, is shared (top violet part in Fig. 1a). There is no class overlap among different tasks, and accuracy is computed separately for each task. This is done by creating a leaf branch for each task, called head, which classifies only images associated to that task (right part in Fig. 1a). Mathematically, we have a model $f(\mathbf{x}, t) = S_t(E(\mathbf{x}))$, where $E(\cdot)$ is the encoder and $S_t(\cdot)$ a task-specific classifier. In a naive setting, every time we receive a new task t, we minimize a task-specific loss starting from the current encoder E and the new randomly-initialized S_t:

$$\mathcal{L}_t = \sum_{i=1}^{S} L(f(\mathbf{x}_i, t), y_i)\,, \tag{1}$$

where L is a suitable loss (e.g., cross-entropy). CF appears whenever training on the current task t degrades the performances on previous tasks $1, \dots, t-1$.

Such models cannot be used to classify unknown samples, since the belonging task needs to be known a priori. These experiments are suitable for studying the feasibility of training disjoints tasks without forgetting how to solve the previous ones. We leave explorations of alternative scenarios, such as the so-called Single Incremental Task (SIT) (Maltoni & Lomonaco, 2019), for future work.

3.3. Pseudo-rehearsal in the embedding space

As shown in Fig. 1a, we augment the classifier described in Section 3.2 with two additional networks:

1. Firstly, we add a decoder network $\hat{\mathbf{x}} = D(\mathbf{z})$ to approximately invert the encoder network. Both of the networks are trained jointly as an autoencoder network.

2. Secondly, we train a generative model to generate samples from $p(\mathbf{z} \,|\, t, y)$, i.e., embeddings of a specific task and class. This model is implemented with a NF, described in Section 3.4.

Given the data for the new task, the training process is divided in three phases:

1) Autoencoder training: In the first stage we train the encoder and decoder networks on the current task, by minimizing a reconstruction loss:

$$\mathcal{L}_{ae} = \sum_{i=1}^{S} \|\mathbf{x}_i - D(E(\mathbf{x}_i))\|^2\,. \tag{2}$$

To avoid to use all the capacity of the autoecnoder we implement it as a sparse autoencoder (Arpit et al., 2016). In this way, by regularizing the layers of our network, not only the reconstruction improves but also the classification scores, due to the sparsity of the produced embeddings.

1.1) Pseudo-rehearsal embedding regularization

If the task is not the first one, we further overwrite a percentage of each mini-batch with samples drawn from the generative model. We sample from the generative model using the same proportions of labels observed in the training sets. In addition to using the generative model to augment the dataset, we also apply a regularization technique similar to the ER method proposed in (Pomponi et al., 2020). This step is applied only while we train the autoencoder and only its weights are updated; this step is is depicted in Fig. 1b.

Consider a generic sample $\hat{\mathbf{z}} \sim p(\mathbf{z} \,|\, t, y)$ from the generative model. We first project the embedding to the input space using the trained decoder $\hat{\mathbf{x}} = D(\hat{\mathbf{z}})$. Then, we regularize the autoencoder by penalizing deviations from the currently generated embeddings:

$$R(\hat{\mathbf{z}}) = d(\hat{\mathbf{z}}, E(D(\hat{\mathbf{z}})))\,, \tag{3}$$

where d is a distance function. Practically, when training the autoencoder, we augment the reconstruction loss in (2) by computing (3) on all the data we use to augment the dataset.

2) Generative model training: In the second stage, we train the generative model to accurately sample embeddings from the new task t. This procedure is described in Section 3.4. We underline that we use a single generative model conditioned on the task, in order to keep the memory overhead of our method constant. As in the autoencoder training, if the task is not the first one, we overwrite a percentage of each mini-batch with samples associated from the past tasks, drawn from the generative model. In this phase, regularize the embeddings is not needed, since the NF is will be trained on new and past embeddings jointly; the training itself acts like a consolidation of past information.

3) Classifier training: In the third stage, the current task-specific head S_t is trained to solve the current task t by minimizing (1).

We found that splitting the training into three separate processes helps the stability and improves the results, since each

block works with the others already optimized. The computational overhead is also small, since each step updates a small part of the overall architecture (we provide memory and time comparisons in our experimental evaluation).

3.4. Normalizing Flows

To complete the specification of our PRER method, we need to describe a specific generative model to estimate samples from the embedding space. While most generative methods could be used here, we have found normalizing flows (NFs) to be particularly effective for the task. Introduced in (Tabak & Vanden-Eijnden, 2010) and (Tabak & Turner, 2012), and popularized by (Rezende & Mohamed, 2015) and (Dinh et al., 2014), NFs are probabilistic models describing the transformation of a probability distribution into a more complex one using a sequence of differentiable, invertible mapping functions. Depending on the type, a NF is capable of efficiently performing sampling or density estimation in either direction (Papamakarios et al., 2019).

Ignoring for a moment the conditioning on a given task, let $\mathbf{u} \sim p_u(\mathbf{u})$ be a real d-dimensional distribution which is easy to sample from, and T a transformation $T : \mathbb{R}^d \rightarrow \mathbb{R}^d$. The key requirements for T to be a NF are: 1) T must be invertible, with an inverse denoted by T^{-1}, and 2) both T and T^{-1} must be differentiable. We can obtain samples from $p_z(\mathbf{z})$ by drawing samples from the easier $p_u(\mathbf{u})$ and then computing $T(\mathbf{u})$ (forward mapping). Alternatively, we can 'normalize' a known sample $\mathbf{z} \sim p_z(\mathbf{z})$ by applying the inverse transformation $T^{-1}(\mathbf{z})$ (inverse mapping).[1] Additionally, we can evaluate the likelihood of a known sample $\mathbf{z}$ as:

$$p_z(\mathbf{z}) = p_u(\mathbf{u}) \Big| \det J_T(T^{-1}(\mathbf{x})) \Big|^{-1}, \tag{4}$$

where $J_{T^{-1}} \in \mathbb{R}^{d \times d}$ is the Jacobian matrix of all partial derivatives of T^{-1}. The prior distribution $p_u(\mathbf{u})$ is generally chosen as an Isotropic Normal distribution. T is instead obtained by composing multiple simpler, invertible transformations $T_1, \ldots, T_L$, resulting in $T = T_L \circ T_{L-1} \circ \cdots \circ T_1$. The arbitrarily complex density $\mathbf{z}$ can be constructed from a prior distribution by composing several simple maps and then applying Eq. 4.

As stated before, a NF can perform both sampling and density estimation in both directions. Depending on the specific NF, not all of these operations are necessarily easy to compute (Papamakarios et al., 2019). This is due to the fact that to achieve them simultaneously, the model needs both a simple forward mapping, a simple inverse mapping, and the Jacobians have to be easy to compute. In general, this is not always possible, mostly due to a computational trade-off. Hence, a NF model needs to be built depending on the application.

In our case, we need efficient density estimation and sampling in the inverse mapping (for training), but only efficient sampling in the forward direction (see Fig. 1b). We use a setup similar to (Dinh et al., 2016; Kingma & Dhariwal, 2018), whose layers we briefly summarize below. Denote by $\mathbf{u}_{j+1} = T_j(\mathbf{u})$ the generic jth block of the NF. We build the overall NF by interleaving three types of invertible transformations.

1) Coupling Layer and Affine transformation: Proposed in (Dinh et al., 2014; 2016), a coupling layer consists in a powerful reversible transformation, where the forward and the inverse mappings are computationally efficient. Here, we use the version presented in (Kingma & Dhariwal, 2018). Consider a generic split of the input vector $\mathbf{u}_j = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}$. A coupling layer is defined as:

$$\begin{aligned} &\log \mathbf{s}, \mathbf{t} = f_\theta(\mathbf{a}) \\ &\mathbf{c_b} = \exp(\log \mathbf{s}) \odot \mathbf{b} + \mathbf{t} \\ &\mathbf{u}_{j+1} = \text{concat}(\mathbf{a}, \mathbf{c_b}) \end{aligned}$$

where $f_\theta(\cdot)$ is a generic transformation implemented via a NN, and $\odot$ is the element-wise multiplication. The $\log \det$ of a coupling layer is simply $\sum_i \log s_i$. The main advantage of this layer is that the function $f_\theta(\cdot)$ does not have to be invertible. The main advantage is also a disadvantage, since, due to the simplicity of the affine transformation, a NF implemented with this technique needs multiple layers and blocks in order to have enough expressive power to transform any input into $p_u(\mathbf{u})$.

2) Random Permutation: since only a portion of the input is modified in each block, it is required to randomly permute the output of a block in order to modify a new set of parameters in the following block, i.e., $T_j(\mathbf{u}) = \mathbf{P}\mathbf{u}$, where $\mathbf{P}$ is a fixed permutation matrix. Clearly, $T_j^{-1}(\mathbf{u}_{j+1}) = \mathbf{P}^T \mathbf{u}_{j+1}$, and the transformation has a unitary determinant.

3) Invertible Batch Norm: As in (Dinh et al., 2016), we also apply batch normalization, but on the input of the coupling layer instead of the output. It acts as a linear re-scaling of each parameter, thus it can be easily inverted and included in the Jacobian computation. The scaling is done in the following way:

$$\mathbf{u}_{j+1} = \frac{\mathbf{u} - \boldsymbol{\mu}}{(\sqrt{\boldsymbol{\sigma}^2 + \epsilon})}$$

and the parameters are iteratively estimated as:

$$\begin{aligned} \boldsymbol{\mu} &= m\boldsymbol{\mu} + (1-m)\boldsymbol{\mu}_b \\ \boldsymbol{\sigma} &= m\boldsymbol{\sigma} + (1-m)\boldsymbol{\sigma}_b \end{aligned}$$

[1] As mentioned in (Papamakarios et al., 2019), the terms "forward" and "inverse" are simply a convention.

where $m \in [0, 1]$ is the momentum, while $\boldsymbol{\mu}_b$ and $\boldsymbol{\sigma}_b$ are, respectively, the mean and the standard deviation of the current mini-batch. On the first batch, the parameters are initialized as $\boldsymbol{\mu} = \boldsymbol{\mu}_b$ and $\boldsymbol{\sigma} = \boldsymbol{\sigma}_b$, and they are updated with each new batch during the training. The $\log\det$ is computed as:

$$\left(\prod(\boldsymbol{\sigma}^2 + \epsilon)\right)^{-0.5}$$

with $\epsilon > 0$ a parameter to avoid zero multiplication.

Note that, in our case, we need a NF that is conditioned on the pair (t, y). To implement this, we condition the first coupling layer by passing the pair (t, y) as an additional argument to f_θ, in order to be able to generate embeddings for every possible class. The idea is similar to what is done in (Winkler et al., 2019). Practically, the conditioning is done by building a one-hot vector with respect to the maximum number of separate classes we presume to observe. We have found the NF to be very easy to train and condition in all our experiments, with no need for specific techniques to avoid CF. In Fig. 2 we show some examples of generated embeddings from the MNIST dataset, using several conditioning vectors.

Furthermore, we implement a multi scale architecture as explained in (Dinh et al., 2016; Kingma & Dhariwal, 2018). Each mapping T_j is decomposed into multiple sub-mappings. After each sub-level, the output is split in two, with the first part being sent directly to the output of the level, while the other part flows into the next sub-mapping for further processing. This architecture improves the convergence and the stability, and it results in a smaller number of parameters.

Many others way of building a NF exists; for an in-depth review of the NF literature we refer to (Papamakarios et al., 2019) and (Kobyzev et al., 2020).

4. Experiments

4.1. Datasets and metrics

For evaluating the proposed method, we consider three different datasets: MNIST, KMNIST (Clanuwat et al., 2018), SVHN (Netzer et al., 2011) and CIFAR10 (Krizhevsky, 2009). We evaluate the methods under the previously described multi-head CL scenario; to do so, being C the classes of a dataset, these are grouped in M sets, each one making up a task containing $c_m \in \mathbb{N}_+$ classes, with $c_m > 1$, giving: $M = \frac{C}{c_m}$ (with this formulation each task contains the same number of classes, with the exception of the last one if $C \bmod c_m \neq 0$). In particular, we split the labels by grouping the original ones in an incremental way.

To evaluate the efficiency and to compare the methods two metrics from (Díaz-Rodríguez et al., 2018) have been used.

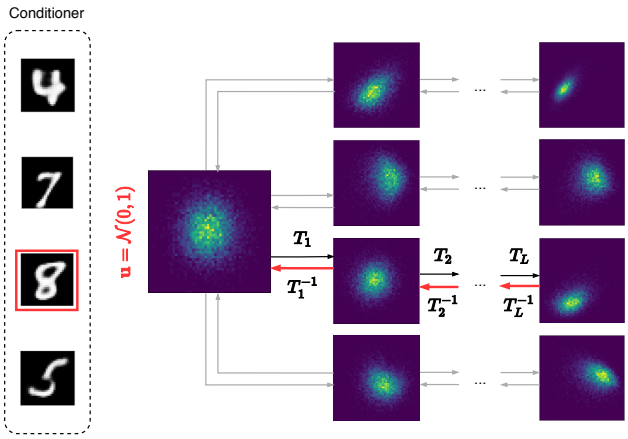


Figure 2. Visualization of the conditional embeddings produced by the trained NF on the MNIST dataset. We use PCA for visualization.

All these metrics are calculated on a matrix $R \in \mathbb{R}^{M \times M}$, where M is the number of tasks, and each entry R_{ij} is the test accuracy on task j when the training on task i is completed. This matrix of scores can be used to calculate different metrics, and we use the following:

Accuracy: it is the average accuracy on the trained tasks. It considers the elements of the diagonal as well as the elements below it, doing so also the evolution of the scores is taken into consideration:

$$\text{Accuracy} = \frac{\sum_{i>j}^{M} R_{ij}}{\frac{1}{2}M(M+1)}.$$

This metric aims to show the average performance of the model in every step of the training and for each task.

Backward Transfer (BWT): it measures how much information from the old tasks is remembered during the training on the new one. It is calculated as:

$$\text{BWT} = \frac{\sum_{i=2}^{M}\sum_{j=1}^{i-1}(R_{ij} - R_{jj})}{\frac{1}{2}M(M-1)}.$$

This metric can be greater than zero, meaning that not only the model is remembering everything about the past tasks, but it also improves the score on these. In our scenario, this phenomenon is rare, since heads associated to past tasks are no longer trained.

We use these metrics because they embed all the important aspects of the CL problem: the ability of an approach to mitigate CF and to classify correctly the past tasks, but also the ability to train the model on the current task. They are both important because a model with high accuracy and low BWT is a model not capable of alleviating CF; on the

Table 1. Average percentage on 5 runs, and the associated standard deviation, for BTW and accuracy, obtained with each method and evaluated dataset. All the results are calculated on the test sets. Best results within standard deviation are reported in **bold**.

	MNIST		KMNIST		SVHN		CIFAR10	
	BWT	ACCURACY	BWT	ACCURACY	BWT	ACCURACY	BWT	ACCURACY
NAIVE	$-10.09_{\pm 2.46}$	$92.95_{\pm 1.62}$	$-5.09_{\pm 1.46}$	$95.95_{\pm 1.04}$	$-12.61_{\pm 1.12}$	$84.81_{\pm 0.80}$	$-24.33_{\pm 5.51}$	$71.28_{\pm 3.66}$
LWF	$-7.17_{\pm 1.56}$	$94.94_{\pm 1.04}$	$-4.30_{\pm 3.64}$	$94.44_{\pm 1.89}$	$-12.19_{\pm 0.73}$	$88.74_{\pm 0.48}$	$-24.77_{\pm 0.94}$	$70.71_{\pm 0.30}$
EWC	$-7.89_{\pm 1.44}$	$94.84_{\pm 0.95}$	$-3.60_{\pm 3.45}$	$95.53_{\pm 2.36}$	$-10.77_{\pm 1.10}$	$89.32_{\pm 0.73}$	$-23.95_{\pm 4.12}$	$71.65_{\pm 4.12}$
GEM	$-0.60_{\pm 0.24}$	$99.10_{\pm 0.16}$	$-1.62_{\pm 0.46}$	$97.77_{\pm 0.44}$	$-2.41_{\pm 0.32}$	$\mathbf{95.22_{\pm 0.63}}$	$-5.50_{\pm 0.51}$	$\mathbf{82.69_{\pm 0.69}}$
ER	$-0.36_{\pm 0.47}$	$\mathbf{99.39_{\pm 0.48}}$	$-0.73_{\pm 0.30}$	$\mathbf{98.68_{\pm 0.29}}$	$-1.63_{\pm 0.07}$	$\mathbf{95.32_{\pm 0.06}}$	$-4.73_{\pm 0.63}$	$\mathbf{82.22_{\pm 0.73}}$
PRER	$\mathbf{-0.14_{\pm 0.01}}$	$\mathbf{99.41_{\pm 0.10}}$	$\mathbf{-0.26_{\pm 0.09}}$	$\mathbf{98.70_{\pm 0.20}}$	$\mathbf{-0.46_{\pm 0.17}}$	$\mathbf{95.23_{\pm 0.06}}$	$\mathbf{-3.63_{\pm 0.64}}$	$\mathbf{82.67_{\pm 0.94}}$

other hand, low accuracy and high BWT tells us that the constraints applied on the model are too restrictive, blocking the training of the current task.

4.2. Experimental setting

We compare our method to the naive approach –training on all the tasks sequentially without mitigating the CF problem– and to the most popular or related rehearsal and regularization approaches: LWF, EWC, GEM, and ER.

With regard to LWF, since the original method was designed for a 2-task scenario and it is not competitive outside it, we modified the approach by transforming it into a rehearsal approach: we have an external memory which contains past samples and the associated past predictions. When regularizing, the current predictions on past tasks' samples are forced to match the past prediction; in this way, this approach is competitive with the other baseline methods.

For MNIST and KMNIST we train the models using SGD with a learning rate equal to 0.1, while for SVHN and CIFAR we use Adam (Kingma & Ba, 2014) with a learning rate equal to 0.001. Regarding PRER, we train both the autoencoders and the NF using Adam, with, respectively, learning rate equal to $1\mathrm{e}{-3}$ and $1\mathrm{e}{-4}$. We use batch size equal to 64 for all datasets with the exception of CIFAR10, for which we used 128 as batch size. Each dataset have been splitted in 5 tasks by setting $c_m = 2$

We fine-tune all hyper-parameters based on a grid-search and the results from the original papers. For PRER we use 2 levels, each one composed by 10 blocks (each block is composed by: batch normalization, a coupling layer, and a random permutation); each coupling layer is defined as a fully connected network with 3 layers: $T : \mathbb{R}^{\frac{d}{2}} \rightarrow \mathbb{R}^d \rightarrow \mathbb{R}^{\frac{d}{2}}$. We also find that setting the coupling layers scale as $\tilde{s} = \text{sigmoid}(s + 2)$, so that the initial scale is near the identity, helps in stabilizing the training phase.

The encoder is composed by three convolutional layers of 12, 24, and 48 filters for MNIST and KMNIST, and twice these sizes for SVHN and CIFAR10. Each layer has 4×4 kernel sizes and stride 2×2. Then, the output of the last convolutional layer is flattened and, using a fully-connected layer, the final embedding vector is produced: for MNIST and KMNIST we use 50 as embedding size, while we use 100 for SVHN and 200 for CIFAR10. The head takes as input the embedding vector, and predicts the class using three fully-connected layers, each of which halves the dimension of its input; we also use a dropout layer, with probability set to 0.2, between consecutive layers. The activation function used in all the models is the ReLU.

Table 2. Approximate memory requirements, without taking into account the base network, and training time to do a complete training on MNIST and CIFAR10 are reported. The memory column contains, for each method, the required space (counted as number of floats to store); in order to calculate the required space we define: N is the dimension of the encoder, M is the number of tasks, c_m is the number of classes per task, IM is the dimension of an image, E is the dimension of the embedding extracted by the encoder, S is the number of samples saved in the external memory, and NF is the dimension of the pair (NF, decoder) used in our method.

		MNIST		CIFAR10	
	Memory	Memory	Time (m)	Memory	Time (m)
Naive	-	-	≈ 9	-	≈ 20
LWF	$M \times S \times (c_m + IM)$	$\approx 1960K$	≈ 20	$\approx 7680K$	≈ 30
EWC	$M \times N$	$\approx 310K$	≈ 12	$\approx 2742K$	≈ 25
GEM	$M \times S \times IM$	$\approx 1960K$	≈ 60	$\approx 7680K$	≈ 70
ER	$M \times S \times (E + IM)$	$\approx 834K$	≈ 15	$\approx 8180K$	≈ 40
PRER	NF $+N$	$\approx 148K$	≈ 20	$\approx 1279K$	≈ 50

We repeat each experiment for 5 times, each time changing the split of the dataset (balanced split based on labels, with proportion 80% train and 20% test) as well as the initial weights of the models.

4.3. Results

In Table 1 we summarize the results concerning accuracy and BTW, while Table 2 shows the required time and memory space for each method. Results in Table 2 are shown for brevity only on MNIST, but they are similar on the other two datasets.

First, we look to the baseline methods. We can see that, in this multi-head scenario, all methods perform better that the Naive approach. LWF and EWC perform better than the naive method but not by much, and they are also more sensi-

tive to the starting weights (as can be seen by the variance) and, in the case of LWF, also to the images stored in the external memory; furthermore, LWF performance degrades when the dataset becomes more complex, which leads us to conclude that it is not a reliable method for alleviating the CF problem. The others baseline methods, GEM and ER, performs better than LWF and EWC, with ER being slightly better in both BWT and accuracy. If we consider also the required time and the memory requirements, which are summarized in Table 2, we can conclude that ER is the best baseline method, because it requires less memory and less time than GEM (as advocated in (Pomponi et al., 2020)).

The method we propose, PRER, achieves better BWT score than all the others methods (by a significant margin), and comparable accuracy to the best one; it is also more robust to the weights' initialization. Looking at the required memory space, PRER achieves the best results w.r.t. the others baseline methods, because, given a NF which is large enough, the generative model is capable of generating all the classes encountered during the training, requiring constant memory. In terms of time, it depends mostly on the dimension of the generative model, which needs to be trained separately: in the MNIST example, shown in Table 2, the required time is competitive to the other methods due to the small NF that works directly on the embeddings.

5. Conclusion

In this paper we introduced PRER (Pseudo-Rehearsal Embedding Regularization), a pseudo-rehearsal method that, by working on the embedding space, is able to generate past embeddings and use them to protect past information while learning new tasks. This approach is a different point of view on the pseudo-rehearsal methods, which usually work on the input space. By working on a lower complexity space, the required time is reduced as well as the dimension of the generative model. Once combined with a decoder, the generative model can be used to generate images associated to past tasks, and to constrain the model by regularizing the embeddings' deviation. We believe that this set of methods can be further investigated, leading to a different view of the pseudo-rehearsal approaches, which, right now, are feasible only for low complexity datasets. We leave an investigation on how the PRER method scales to more complex datasets to future work.

References

Aljundi, R., Kelchtermans, K., and Tuytelaars, T. Task-free continual learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11246–11255, 2019.

Arpit, D., Zhou, Y., Ngo, H., and Govindaraju, V. Why regularized auto-encoders learn sparse representation? In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 136–144. PMLR, 20–22 Jun 2016.

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

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Díaz-Rodríguez, N., Lomonaco, V., Filliat, D., and Maltoni, D. Don't forget, there is more than forgetting: new metrics for continual learning. *arXiv preprint arXiv:1810.13166*, 2018.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

Ebrahimi, S., Meier, F., Calandra, R., Darrell, T., and Rohrbach, M. Adversarial continual learning. *arXiv preprint arXiv:2003.09553*, 2020.

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

Guo, Y., Chen, Q., Chen, J., Wu, Q., Shi, Q., and Tan, M. Auto-embedding generative adversarial networks for high resolution image synthesis. *IEEE Transactions on Multimedia*, 21(11):2726–2737, 2019.

Kang, W., Cheol-Ho, H., and Byoung-Tak, Z. Discriminative variational autoencoder for continual learning with generative replay, 2020.

Kingma, D. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10215–10224. Curran Associates, Inc., 2018.

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

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pp. 4743–4751, 2016.

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

Kobyzev, I., Prince, S., and Brubaker, M. A. Normalizing Flows: an introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

Krizhevsky, A. Learning multiple layers of features from tiny images, 2009.

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

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Maltoni, D. and Lomonaco, V. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116: 56–73, 2019.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In Bower, G. H. (ed.), *Psychology of Learning and Motivation*, volume 24, pp. 109 – 165. Academic Press, 1989.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54 – 71, 2019.

Pomponi, J., Scardapane, S., Lomonaco, V., and Uncini, A. Efficient continual learning in neural networks with embedding regularization. *Neurocomputing*, 397:139 – 148, 2020.

Rao, D., Visin, F., Rusu, A., Pascanu, R., Teh, Y., and Hadsell, R. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems*, pp. 7645–7655. Curran Associates, Inc., 2019.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538. PMLR, 07–09 Jul 2015.

Robins, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

Roger, R. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, 97 2:285–308, 1990.

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

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 4548–4557. PMLR, 2018.

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

Tabak, E. G. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2012.

Tabak, E. G. and Vanden-Eijnden, E. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Thrun, S. and Mitchell, T. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25 – 46, 1995.

Winkler, C., Worrall, D., Hoogeboom, E., and Welling, M. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

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

Zeno, C., Golan, I., Hoffer, E., and Soudry, D. Task agnostic continual learning using online variational Bayes. *arXiv preprint arXiv:1803.10123*, 2018.