

# Efficient Continual Learning in Neural Networks with Embedding Regularization

Jary Pomponi<sup>b</sup>, Simone Scardapane<sup>b,\*</sup>, Vincenzo Lomonaco<sup>a</sup>, Aurelio Uncini<sup>b</sup>

<sup>a</sup>*Department of Computer Science and Engineering, University of Bologna, Italy*

<sup>b</sup>*Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Italy*

---

## Abstract

Continual learning of deep neural networks is a key requirement for scaling them up to more complex applicative scenarios and for achieving real lifelong learning of these architectures. Previous approaches to the problem have considered either the progressive increase in the size of the networks, or have tried to regularize the network behavior to equalize it with respect to previously observed tasks. In the latter case, it is essential to understand what type of information best represents this past behavior. Common techniques include regularizing the past outputs, gradients, or individual weights. In this work, we propose a new, relatively simple and efficient method to perform continual learning by regularizing instead the network internal embeddings. To make the approach scalable, we also propose a dynamic sampling strategy to reduce the memory footprint of the required external storage. We show that our method performs favorably with respect to state-of-the-art ap-

---

\*Corresponding author. Phone: +39 06 44585495.

*Email addresses:* `simone.scardapane@uniroma1.it` (Simone Scardapane), `vincenzo.lomonaco@unibo.it` (Vincenzo Lomonaco)

proaches in the literature, while requiring significantly less space in memory and computational time. In addition, inspired by recent works, we evaluate the impact of selecting a more flexible model for the activation functions inside the network, evaluating the impact of catastrophic forgetting on the activation functions themselves.

*Keywords:* Continual learning, catastrophic forgetting, embedding, regularization, trainable activation functions

---

## 1. Introduction

While deep networks have achieved remarkable successes over the last decade, most of their applications are limited to single, isolated problems, where each network has to be redesigned (and re-trained) from scratch. As a result, their training remains daunting in situations where data is scarce and/or computation is expensive. In these scenarios, efficient transfer of information from one task to another (e.g., in reinforcement learning contexts) is widely acknowledged to be a fundamental technique for their deployment [22]. Depending on the number of tasks and their similarity, different applicative scenarios can be devised, ranging from simple transfer learning from one source domain to a target domain (e.g., as is popular today with pre-trained image classification networks), up to full, *lifelong* learning of a single network on a possibly infinite number of interrelated tasks.

Artificial general intelligence must be able to learn and remember many different tasks [17] and this is particularly complex in real-world scenarios. The ability of an agent to learn consecutive tasks without forgetting how to solve previously learned tasks is called continual learning (CL). CL is a

property difficult to obtain, because when the network learns how to solve a task the information about how to solve previously learned tasks is generally lost if no countermeasure is adopted; this phenomenon is called *catastrophic forgetting* [8], and it occurs when the network, trained on a task, changes its weights in order to meet the objectives of the new task. As a result, the network accuracy on a previous task is drastically reduced after a few training updates on successive tasks. Whilst recent results on deep networks have improved the results on a wide variety of domains, relatively less progress has been made in reaching continual learning on artificial neural networks.

Because of its importance, many ideas and algorithms were proposed recently for an effective CL of neural architectures. One class of algorithms, known as progressive training algorithms [6, 26, 28], incrementally builds the structure of the network for each task being processed, trying to re-use as much as possible of the previous architecture in the process. Another class of algorithms, known as *rehearsal methods* [24, 29, 2], save the information about past tasks in an external memory, and continually retrain the network on this memory in order to maintain its performance. Finally, a third class of algorithms try to reuse a single neural network by instead including multiple *regularization* penalties [14] to stabilize the behaviour of the network with respect to previous tasks. Generally speaking, rehearsal and progressive strategies tend to perform well but scale poorly with respect to the number of tasks, and might have a large computational complexity. Differently, regularization strategies are simple to implement, tend to require less memory, but their performance might not match naive rehearsal.

One key problem when implementing a regularization strategy is deciding

what information best represents the behaviour of the network and, as a result, what form the regularization penalty should take. Proposals in this sense range from regularizing individual weights based on their importance [14], to specific gradients [20], outputs [18] (similar to the idea of knowledge distillation), using information from an external memory or a combination of these. Some state-of-the-art algorithms in this sense are described in-depth in Section 2. Choosing the proper level of abstraction in this sense is not trivial, and the choice highly influences the level of accuracy of these techniques, and their ability to scale on multiple tasks.

#### *Contributions of this paper*

In this paper, we propose a novel regularization-rehearsal approach for continual learning. Our insight is that an efficient way to regularize the behaviour of the network is to act on its internal *embeddings*, i.e., the activations of one or more layers close to the exit. We argue that regularizing the embeddings provide more information as opposed to simply asking for similar outputs, allowing the network more degrees of freedom in choosing how best to enforce the penalty, while being less stringent than asking for specific, individual weights to be preserved.

Because our method requires a small external memory of past activations to work efficiently, we also devise a simple sampling strategy for this memory to maximize the information provided by the regularization term, i.e., we try to sample maximally those elements in memory that can provide the strongest regularization on the model.

In addition to the previous contributions, we also evaluate the impact of more flexible activation functions, namely, kernel activation functions (KAFs,

[27]), on the CL scenario. Several papers [9, 1, 12, 23, 27] recently have shown that, in the single-task scenario, training the activation functions can provide significant improvements in performance and in the number of layers required to solve the given task. The intuition is that giving more freedom to the activation functions inside the network, whenever these are properly normalized, can have a beneficial impact on the learning of the networks. However, very little has been done to evaluate whether these benefits extend to the CL case or if, on the opposite, flexible activation functions can suffer from catastrophic forgetting themselves.

To this end, in the experimental section we compare our proposed regularization technique with multiple state-of-the-art regularization algorithms for CL, both on standard neural networks and on neural networks endowed with KAFs, hereafter denoted as *Kafnets*.

#### *Organization of this paper*

The rest of the paper is structured as follows. Section 2 describes the problem of CL in neural networks and some techniques for performing CL through proper regularization. Then, after elaborating on the advantages and shortcomings of each of these techniques, in Section 3 we propose a novel method for performing the regularization, that we call embedding regularization (ER). We provide a comprehensive experimental evaluation of the proposed ER algorithm in Sections 4 and 5, before giving some concluding remarks in Section 6.

## 2. Continual learning in neural networks

### 2.1. Problem formulation

We follow the formulation of continual learning from [20]. We receive a (possibly infinite) sequence of labeled examples, and we suppose that each example is described by a triplet  $(x, t, y) \in \mathcal{X} \times \mathbb{N}^+ \times \mathcal{Y}$ , where  $x$  is an input (e.g., an image or a vector of features),  $t$  is an integer identifying a particular task, and  $y$  is the corresponding label of  $x$  (e.g., the presence of a specific object inside the image).

In this formulation,  $x$  can belong to any domain (e.g., an image, or a vector in a suitable Euclidean space), although most benchmarks in the CL literature have considered the image domain [22]. As a motivating example, images with  $t = 0$  could contain either cars or trucks (two classes), while images with  $t = 1$  could contain planes, helicopters, or boats (three classes) and so on. Crucially, we assume that a pair  $(x, y)$  is locally i.i.d. given  $t$ , i.e., it is sampled from some probability distribution  $P_t$  that fully specifies the task. The total number of tasks can be known *a priori* or not.

In addition, several works in the literature consider sequential scenarios where a full dataset belonging to a given task is received and processed (even with multiple passes over it), before switching to the next task. Nonetheless, methods developed for the sequential case can generally be extended to the non-sequential (online) case, and *vice versa*.

In continual learning we want to design a training algorithm for a neural network  $f_\theta(x, t)$ , parameterized by a vector  $\theta$ , such that  $f_\theta(x, t) \approx y$  for any possible task and example. Because we assume potential correlations between different tasks, the network should share as much as possible the

information contained in  $\theta$  across all possible problems.

For a single task  $t$  (e.g., only images of cars/trucks), we could find the optimal  $\theta$  by minimizing the expected risk of the network as:

$$\theta^* = \arg \min_{\theta} \left\{ \mathbb{E}_{(x,y) \sim P_1} \mathcal{L}(f_{\theta}(x, t), y) \right\}, \quad (1)$$

where  $\mathcal{L}(\cdot, \cdot)$  is a loss function and the expectation is taken with respect to all possible data of the single task. If we are provided with a dataset of  $B$  examples  $\{x_i, t, y_i\}_{i=1}^B$  from the task, then (1) can be approximated by minimizing:

$$J_t(\theta) = \left\{ \frac{1}{B} \sum_{i=1}^B \mathcal{L}(f_{\theta}(x_i, t), y_i) \right\}. \quad (2)$$

Practically, (2) can be solved by taking mini-batches of data and using these to compute approximate gradients of the cost function  $J(\theta)$  [3].

Extending this formulation to more than a single task is non-trivial for the problems described in Section 1, mostly because of the catastrophic forgetting of previously learned information [22]. In particular, suppose we are provided with a separate dataset of  $B'$  examples  $\{x_i, t', y_i\}_{i=B+1}^{B+B'}$  from a different task  $t'$  (e.g., a new mini-batch of data containing planes and similar objects). A naive solution would be to solve a problem  $J_{t'}(\theta)$  similar to (2), starting however from the previous solution  $\theta^*$ . Catastrophic forgetting then occurs whenever the solution of this new optimization problem performs poorly on the original task.

Next, we describe many state-of-the-art regularization solutions for approaching this problem. Note how, as the size of the datasets shrink, and as we iterate this process, we are back at the sequential scenario described at

the beginning of this section. Nonetheless, it is easier to describe continual learning techniques in this simplified context.

## 2.2. Regularization by weight consolidation

The idea of elastic weight consolidation (EWC) [14] is to impose a regularization term on successive tasks to preserve as much as possible the weights in  $\theta$  that were essential to the previous task. This is achieved by augmenting  $J_\nu(\theta)$  with a quadratic penalty as follows:

$$R(\theta) = \frac{\lambda}{2} \cdot \sum_z F_z (\theta_z^* - \theta_z)^2, \quad (3)$$

where  $\lambda$  is a hyper-parameter,  $\theta^*$  is the weight vector after training on the first task, and  $F_z$  represents the  $z$ th diagonal element of the Fisher information matrix:

$$F_z = \frac{1}{B} \sum_{i=1}^B \frac{\partial^2 \log [f_{\theta^*}(x_i, t)]}{\partial \theta_z^*}. \quad (4)$$

EWC can be justified from a Bayesian perspective by assuming that the network posterior distribution can be fully specified by the mean vector  $\theta^*$  and a diagonal precision matrix given by the elements  $\{F_z\}_z$ . When training on more than two tasks, different penalties can be combined by storing the mean vectors and Fisher diagonal matrices for each task [14], or a single composite penalty can be obtained in various ways [19, 11, 15].

More in general, when faced with a sequential scenario, we would generate a sequence of weight vectors  $\theta^1, \theta^2, \dots$  and corresponding diagonal Fisher matrices  $F^1, F^2, \dots$ . At a generic iteration  $n$ , a simple online extension of

EWC [28] computes a single dynamic penalty by keeping the last mean vector  $\theta^{n-1}$  and updating the Fisher information matrix as:

$$F^n = F^n + \gamma F^{n-1}, \quad (5)$$

where  $\gamma \in [0, 1]$  is a discounting factor. In the online version of the approach only the optimal weights from the last task are maintained into memory. Alternatively, one can keep an external memory  $\mathcal{M}$  of samples from previous tasks, and use this memory to build an aggregated Fisher information matrix for every iteration. However, how best to sample from previous tasks remains a significant open challenge [14].

### 2.3. Regularization by knowledge distillation

While EWC has been shown to work well in some scenarios, regularizing individual weights might not always work optimally, especially when different penalties start to exert conflicting influences. Learning without forgetting (LWF) [18] is another method for continual learning whose idea is to simply preserve the outputs of a network on older tasks, while leaving it free to modify weights as necessary to achieve this objective. To do so, LWF employs a variant of knowledge distillation [10].

In our notation,  $f_{\theta^*}(x, t')$  is the output of the network using the previous weights (at iteration  $n - 1$ ) on an example from the separate task  $t'$ . Suppose we are in a binary classification task, in which case the output is the probability of  $x$  belonging to the class specified by  $t'$ . To regularize the network, we add a penalty term to  $J_t(\theta)$  forcing the new probabilities to be similar to the old probabilities using a cross-entropy term:

$$R(\theta) = -\lambda \cdot \frac{1}{B'} \sum_{i=B+1}^{B'} f_{\theta^*}(x_i, t') \log [f_{\theta}(x_i, t')] , \quad (6)$$

where  $\lambda$  has the same meaning as in the previous section. For regression problems one can use the mean-squared error, while for multi-class classification problems it is possible to improve on this basic scheme by re-weighting the probabilities using a new temperature hyper-parameter [18]. LWF has the advantage that the training process can potentially discover new weight configurations able to preserve older behaviors.

#### 2.4. Synaptic Intelligence

Synaptic Intelligence (SI), introduced in [32], is a variant of EWC. The main idea is to calculate the weight importance in an online fashion, instead of calculating the Fisher matrix. To do it the authors characterized the trajectory of the network parameters over the optimization process to evaluate the importance of a parameter’s change in the loss’ decrease. In particular, a weight importance w.r.t. optimization can be defined as:

$$F_z = \frac{\Delta J_z}{(\Delta_z)^2 + \xi} \quad (7)$$

where  $\Delta J_z$  is the sum of instantaneous changes to the parameter  $\theta_z$  along the optimization trajectory of  $J_t$  (see [32] for the full definition),  $\Delta_z$  is its absolute variation from the starting condition, and  $\xi$  a small constant to bound the expression in case of numerical issues.

The main advantage of this approach is the possibility to calculate  $F_z$  using the information that are available during the training and no extra

computation is needed. The magnitude of  $F$  can be scaled by proper setting of a scale factor  $1 \geq c \geq 0$ , which regularizes how much of the past information needs to be saved. This factor can then be used in a EWC-like regularization term replacing the Fisher information matrix.

### 2.5. Gradient episodic memory

Gradient episodic memory (GEM) [20] is another technique which regularizes a model by working neither on individual weights nor on the outputs, but on the *gradients* of the model. In GEM, a subset of past tasks is saved into an external fixed size memory, and used to constrain the current gradients to avoid the increasing of losses associated to past tasks, but at the same time allowing their decreasing; this could lead to positive transfer of knowledge to past tasks.

To this end, denote by  $g_t$  the gradient of the loss obtained with a sample from the past task  $t$  using the current network, and by  $g_{t'}$  the gradient vector on the current task, the method alleviates catastrophic forgetting by enforcing the gradients to point in the same direction:

$$\langle g_{t'}, g_t \rangle \geq 0,$$

if the inequality constraints are satisfied, then the proposed parameter update is unlikely to increase the loss on previous tasks. In this case the past information is preserved. Otherwise there is at least one previous task that would experience an increase in loss after the parameter update.

In order to enforce the previous penalty, at every optimization step a

quadratic program with inequality constraints is solved:

$$\begin{aligned} & \underset{v}{\text{minimize}} && \frac{1}{2}v^T G_t G_t^T v + g_{t'}^T G_t^T v \\ & \text{subject to} && v \geq 0 \end{aligned}$$

where  $G_t$  is a matrix containing all the gradients associated to the past task samples (i.e., each row is the gradient associated to one of the  $B$  samples). The new gradients are calculated as  $\bar{g} = G_t^T v^* + g_{t'}$ , where  $v^*$  is the solution to the QP. Basically, the method forces the new gradients to go in the same direction of the gradients associated to past tasks. This method requires more memory than a simpler regularization approach such as EWC, and more computational time, but could provide improved performance.

Recently a new version of GEM, called A-GEM, has been proposed in [5]. This approach tries to ensure, using a different QP w.r.t. GEM, that at every training step the average episodic memory loss over the previous tasks does not increase; the resulting QP used in A-GEM is faster and easier to minimize. The main difference between A-GEM and GEM is that GEM has better guarantees in terms of worst-case forgetting of each task since it prohibits an increase of any past task specific loss, while A-GEM has better guarantees in term of average accuracy obtained, since GEM can prevent a gradient update on the current task in order to satisfy the constraints.

### 3. Proposed embedding regularization (ER) for continual learning

All methods described in the previous section are similar, in the sense that they try to preserve the behaviour of the neural network on previous tasks by modifying the (unconstrained) descent direction in a direction that

alleviates catastrophic forgetting. They differ, however, in what information is the best representation of such behavior: individual weights for EWC and SI, gradient information for GEM, or actual outputs for LWF.

As we stated in Section 1, our aim in this paper is to complement these methods with a separate regularization technique acting on the *embeddings* of the neural network, i.e., the activations of the second-to-last layer of the model. We argue this is both more flexible than individual weights, and more informative than the outputs themselves, as evidenced by the large literature on using pre-trained networks for a variety of fine-tuning tasks and few-shot learning [21]. The method we propose is connected to the less-forgetful learning (LFL) [13] method in domain adaptation, and in a lesser form to neural graph machines [4] for graph-based learning. LFL is not suitable for a continual learning scenario, since the idea is that, given a source network trained on a source domain, the weights of the source network are used as the initial weights of the target network and then the parameters associated to the softmax layer are frozen; this network is then trained. However, while LFL also regularizes a network on a new domain using embeddings, it is not designed to work on more than two tasks, and its memory requirements scale linearly with the number of tasks, since it requires a network for each one. Also, the features regularization is done by minimizing the distance between the features associated to the new task (e.g., the target domain) extracted using the old network and the new one; so the domains need to be correlated somehow, also because the number of classes remains unchanged from one network to another and the softmax layer is fixed. Conversely, we are interested in the more general CL scenario described above, and in

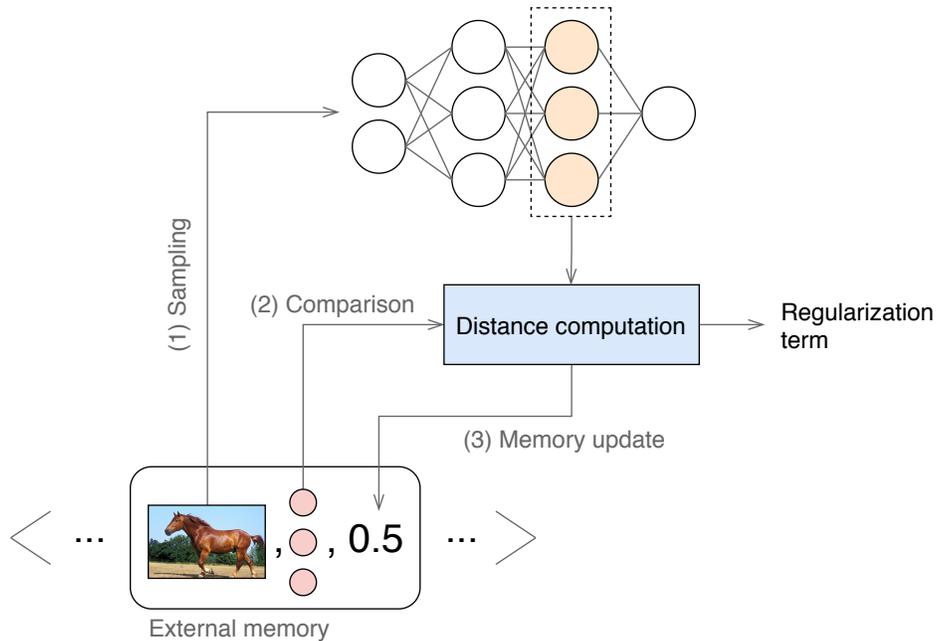


Figure 1: Overall schema of the proposed framework. See the text for a description of the different steps.

designing a method with a fixed, low-memory overhead.

### 3.1. General description of the proposed method

The overall schema of the proposed embedding regularization (ER) technique is provided in Fig. 1. Similarly to before, we represent the continual learning scenario with a small external memory  $\mathcal{M} = \{(x_m, t_m, h_m, p_m)\}_{m=1}^M$ , whose size  $M$  is selected by the user, and where each element of the memory is a quadruplet containing:

- An image  $x_m$  that the network processed previously and the corresponding task  $t_m$ ;

- The embedding  $h_m$  corresponding to the activation of the second-to-last layer of  $f_\theta(x_m, t_m)$ , computed when the network was trained last on task  $t_m$ ;
- A sampling weight  $p_m$ . The set of weights  $\{p_m\}_{m=1}^M$  should define a probability distribution over the elements of  $\mathcal{M}$ .

Whenever we want to regularize our model, we sample an element  $(x_m, t_m, h_m)$  from the past task memory  $\mathcal{M}$  (or a mini-batch of elements) with a probability proportional to  $p_m$ . We then compare the embedding with the current embedding  $\hat{h}_m$  computed with the current weights of the network:

$$R(\theta) = \lambda \cdot d(h_m, \hat{h}_m), \quad (8)$$

where we use the cosine similarity between the two vectors as distance function  $d(\cdot, \cdot)$  in our implementation. The overall algorithm is summarized in Algorithm 1, while in the following we elaborate more in-depth with respect to the sampling strategies we can adopt.

### 3.2. Selecting and updating the sampling probabilities

A correct sampling from the external memory is the key of this method, since sampling unrepresentative images might lead to catastrophic forgetting. The simplest way to do this, besides using uniform sampling probabilities for each value in  $\mathcal{M}$ , is to count how many times a sample  $m_i \in \mathcal{M}$  has already been chosen and assign to the weight a value equal to the inverse of how many times it has been extracted (in the following, this is referred to as the *frequency* sampling strategy). In this way, however, the weights are not a real indicator of the importance of  $m_i$ , given the current task.

---

**Algorithm 1** Embedding regularization (ER)

---

1: **procedure** TRAIN PROCEDURE

2:     *Given:* a neural network  $f$  with current weights  $\theta$

3:     *Given:* a mini-batch of  $B'$  elements from a task  $t$ .

4:     **Step 1** Perform one step of optimization on the mini-batch.

5:     **Step 2** Regularize the model with EmbeddingRegularization().

6:     *When the task is over:* Update the external memory  $\mathcal{M}$ .

7: **procedure** EMBEDDINGREGULARIZATION

8:     Sample  $m$  values  $\{(x_i, t_i, h_i, p_i)\}$  from  $\mathcal{M}$

9:      $m \leftarrow \lambda * \text{mean} \left( \{d(h_i, \hat{h}_i)\}_{i=1\dots m} \right)$

10:     Perform one step of optimization on  $m$ .

---

Another way to do that is to assign an initial value to each weight and then, when  $m_i$  is picked and the distance  $d_i = d(h_i, \hat{h}_i)$  calculated, assign a weight  $p_i$  proportional to the distance  $d_i$  (*distance* sampling strategy). In this way any image that is associated to a high distance will be drawn more frequently, because it contains some information that is in contrast to the current task and needs to be enforced multiple times.

Moreover, the distance between the images can be used, by using the structural distance from an image in the memory and some images from the current task. When a new batch is encountered, an external network, whose parameters are randomly initialized and not trained, is used to extract the features from the images present in the current batch and from the images drawn from the memory (*pretrained reference* sampling strategy). Then the weight  $p_i$  is set equal to the mean distance between the image  $x_i$  from the

memory and all the images in the current batch. The idea is to use a distance that won't change during the training and was inspired by [30].

## 4. Experimental setup

### 4.1. Datasets and metrics

For evaluating the proposed method, we consider the following datasets:

- **Permuted MNIST**: [14] it is a variant of the popular MNIST dataset of handwritten digits [16], in which each task is a random fixed permutation of the pixels and the images are treated as vectors of features (flattened images). In this experiments the output of the network is fixed to 10. This is a relatively easy benchmark, allowing to easily compare with other techniques. In our experiment the number of tasks is set to 4.
- **CIFAR10**: another dataset widely used in the literature. It consists in 10 classes with 60000 colour images each. To adapt this dataset for a continual learning scenario the classes are grouped in  $n$  sets, in our case  $n = 5$ ; so each task contains 2 consecutive classes. In this case the scenario is the CL one.
- **CIFAR100**: a variant of the CIFAR10 dataset with 100 classes. The number of images are the same as CIFAR10 but, since the classes are 100, each class has less images and this makes the training harder. We grouped the classes into 10 tasks.

To evaluate the efficiency and to compare the methods different metrics have been used. All these metrics are calculated on a matrix  $R \in \mathbb{R}^{N \times N}$ ,

where  $N$  is the number of tasks, and each entry  $R_{ij}$  contains the test classification accuracy on task  $j$  when the training on task  $i$  is completed. The metrics used are the following.

**Accuracy:** it is the average accuracy on the trained task, it considers the elements of the diagonal as well as the elements below it:

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

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

**Remembering:** it measures how much information from the old tasks is remembered during the training on the new one. Given a score called backward transfer:

$$\text{backward transfer} = \frac{\sum_{i=2}^N \sum_{j=1}^{i-1} (R_{i,j} - R_{j,j})}{\frac{1}{2}N(N-1)}$$

that measures the influence that learning a task has on the previously learned tasks, the remembering score is defined as:

$$\text{Remembering} = 1 - |\min(0, \text{backward transfer})|.$$

Having a score equals to one means not only that the model is remembering everything about the past tasks, but it can also improve these. To measure how the old tasks have been improved the metric used is **Positive Backward Transfer** =  $\max(0, \text{backward transfer})$ .

We used these metrics because they embody all the import aspects of CL problems: the ability of a model to being able to classify past tasks, by

reducing the CF (**remembering**), but at the same time the ability to correctly classify all the tasks during the training phase (**accuracy**), including the current task. Also, they are jointly important: a model with high accuracy and low remembering is a model not capable of alleviating CF, and will rewrite all the weights at each task; on the other hand low accuracy and high remembering tells us that the constraints applied on the model are too restrictive, blocking the learning of the current task by remembering all the past information.

These metrics and more are defined in [25]. The complete code of our experimental evaluation can be found online.<sup>1</sup>

#### 4.2. Architectures and methods

On the the MNIST experiment a fully-connected neural network with 4 hidden layers of 400 ReLU units is used. This network is trained using SGD with learning rate equal to  $1e - 3$ . For the CIFAR10/100 the CNN is the same used in [31], except for the CIFAR100 case where the number of kernels is doubled. The CNNs have been trained using the Adam optimizer with learning rate set to  $1e - 3$ ; the optimizer state was reset after training on each task.

As stated in the introduction, as a further contribution we also decided to compare the standard NNs to the Kafnets [27], a class of neural networks using KAFs as activation functions. Briefly, using this architecture each activation function is allowed to change shape during training using a small number of adaptable coefficients, and the aim of this set of comparisons is

---

<sup>1</sup> <https://github.com/jaryP/Master-thesis>

to explore whether this additional number of degrees of freedom is beneficial or not in terms of CL. Each Kafnet used has the same architecture of the NN counterpart but the size of each layer/kernel is reduced by 30%, in order to have roughly the same number of adaptable parameters for both architectures. The hyper-parameters for the KAFs use the same values from [27].

To demonstrate the effectiveness of the proposed ER algorithm we compare it with GEM, online EWC, SI, and the full training without any approach to alleviate catastrophic forgetting as further comparisons; we discarded LWF because it showed no good performance in our preliminary experiments on CL (being designed for a single transfer scenario), and also SI from KAFs experiments, given the poor performance obtained on the standard ANN (see below). The memory size for each task in the GEM approach was set to 200 for each task in CIFAR10/MNIST and 500 for each task in CIFAR100. For our method, memory was fixed to a smaller size of 100 images per task. All of the values were fine-tuned to provide the best accuracy, while in Section 5.2 we evaluate the impact of selecting different values for the external memory, for both GEM and ER. For our proposed method, we use the distance sampling strategy in the following, and evaluate the impact of the other sampling strategies described in Section 3.2 in Section 5.3.

Finally,  $\lambda$  was set to 1 in every experiment with the exception of CIFAR100, in which it was set to 5, and the  $c$  factor in SI was set to 1 for permuted MNIST and 0.01 for the other tasks. All of the parameters were chosen after a preliminary empirical analysis.

		Perm-MNIST		CIFAR10		CIFAR100		Average	
		Accuracy	Remembering	Accuracy	Remembering	Accuracy	Remembering	Accuracy	Remembering
Standard NN	None	0.86	0.9509	0.6548	0.6858	0.2487	0.4789	0.5878	0.7052
	EWC	0.8656	0.9894	0.6549	0.7856	0.2187	0.5971	57.94	0.7907
	SI	0.8592	0.9373	0.655	0.6836	0.2532	0.4693	58.91	0.68
	GEM	<b>0.9421</b>	<b>1 (+0.0894)</b>	0.8127	0.8817	0.3211	0.6629	0.6919	<b>0.8511</b>
	ER	0.9128	0.9908	<b>0.8428</b>	<b>0.8905</b>	<b>0.3813</b>	<b>0.6643</b>	<b>71.23</b>	0.8485
Kafnet	None	0.8526	0.6489	0.6620	0.6197	0.2304	0.4441	0.5948	0.5709
	EWC	0.8816	0.9939	0.7107	<b>0.8902</b>	0.2030	0.6030	59.84	0.8290
	GEM	<b>0.912</b>	<b>1 (+0.0157)</b>	0.7320	0.8233	0.2903	0.638	64.47	<b>0.8250</b>
	ER	0.8973	0.9937	<b>0.7856</b>	0.8181	<b>0.3205</b>	0.5975	<b>66.78</b>	0.8030

Table 1: Overall results on the three datasets, two architectures, and different strategies for CL. The plus symbol means that the approach is capable not only of remembering completely the past information about the past tasks, but also to improve the classification of it during the training on new tasks. The average column contains the averaged Accuracy and Remembering obtained on all the benchmarks.

## 5. Experimental evaluation

### 5.1. Overall comparison of the algorithms

Table 1 summarizes all the scores obtained on all datasets, methods and architectures. We decided to use the training without any approach to alleviate catastrophic forgetting as baseline. Overall, the ER method performs similarly or better than GEM, for both accuracy and remembering; with the exception of the MNIST dataset, in which GEM performs better. In general, including more flexible activation functions does not seem to bring benefit to the CL task, worsening the results in most scenarios, possibly because of catastrophic forgetting occurring on the activation functions themselves.

Figs. 2 and 3 show how the score on the first task evolves during the training, respectively on CIFAR10 and CIFAR100. These images show how the proposed ER method is the best one on remembering the first task, with

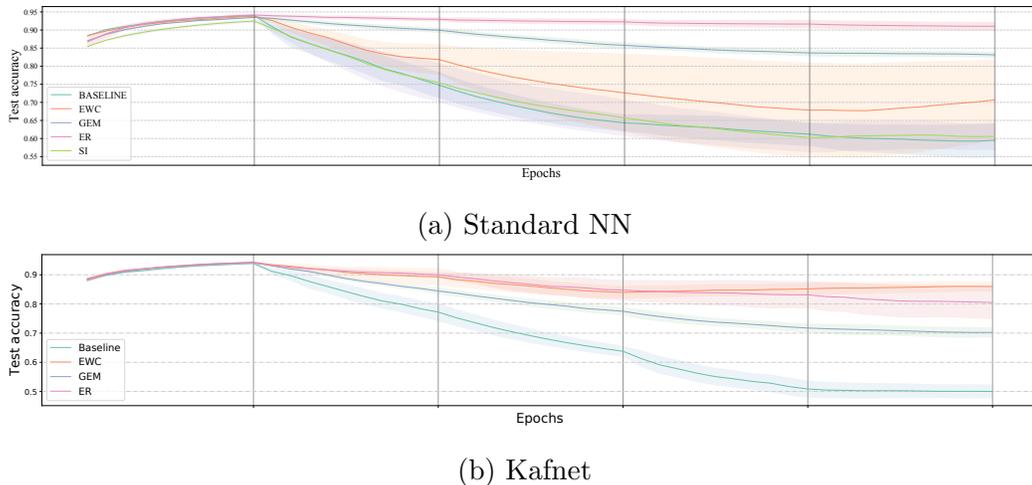


Figure 2: Evolution of accuracy on task 1 when training on CIFAR10.

the exception of the Kafnet on CIFAR10, in which our method achieves better accuracy than GEM but lower remembering. The ER approach achieves better remembering on standard ANN trained on CIFAR10 given the less binding constraints imposed to the networks by the ER, w.r.t. GEM or EWC, because we regularize only the last layer of the network letting the network to adjust the internal weights as it sees fit, giving it more operating space while learning new tasks. This is the most important advantage over the other compared methods.

We complete the analysis in this section with a statistical analysis of the algorithms according to the procedure in Demšar [7]. A Friedman rank test confirms that there are statistical significant differences with respect to the accuracy in Tab. 1 (p-value of 0.01). A successive set of Nemenyi post-hoc tests further confirms statistical significant differences between the proposed method, the baseline network, SI, and EWC.

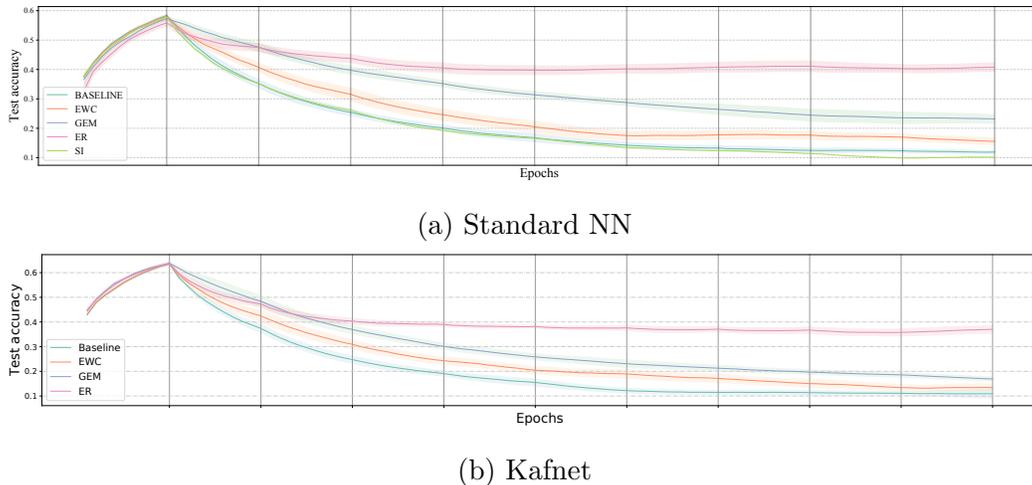


Figure 3: Evolution of accuracy on task 1 when training on CIFAR100.

### 5.2. Impact of the memory size and computational time

The previous section showed that the proposed ER method is able to achieve competing results in terms of accuracy and remembering with a smaller external memory. In this section, we investigate how the performance of the methods varies when we increase or decrease the size of the memory.

To this end, Fig. 4 shows the accuracy and the remembering for both ER and GEM as a function of the memory associated to each task. GEM benefits from a bigger memory while ER achieves the best score around 100.

Training time required by the approaches is shown in Fig. 6. The image shows that, on CIFAR10, ER requires half of the time if compared to GEM, and also ER scales much better when the number of tasks increases (CIFAR100), making GEM not suitable. We can conclude that ER achieves better scores using less memory and the training requires less time, not needing a QP optimization at every optimization step; in fact the QP optimization

problem has a number of parameters equals to the number of tasks, making it slower when the number of tasks in the training grows.

In the experiments about the time we considered also the memory size, but this value is not much impactful if compared to the QP optimization problem; in fact when the memory grows the required training time increases from 10 to 20 seconds, for both GEM and ER.

### *5.3. Impact of the sampling strategy*

Next, we investigate how the different sampling strategies described in Section 3.2 impact the accuracy of the method. Fig. 5 shows the scores obtained when using the different techniques. In general, the distance between the images used as weight performed well, but the real embedding distance achieves better scores.

The image shows also that a sampling approach based on the usage of the images leads to the worst results, since it doesn't provide us any information about the real importance of the images. The distance based sampling method is the best one because it provide useful information about the real distance from the past tasks and the current one, and these distances are adapted during the training, and does not requires any further computation (as opposed to image similarity).

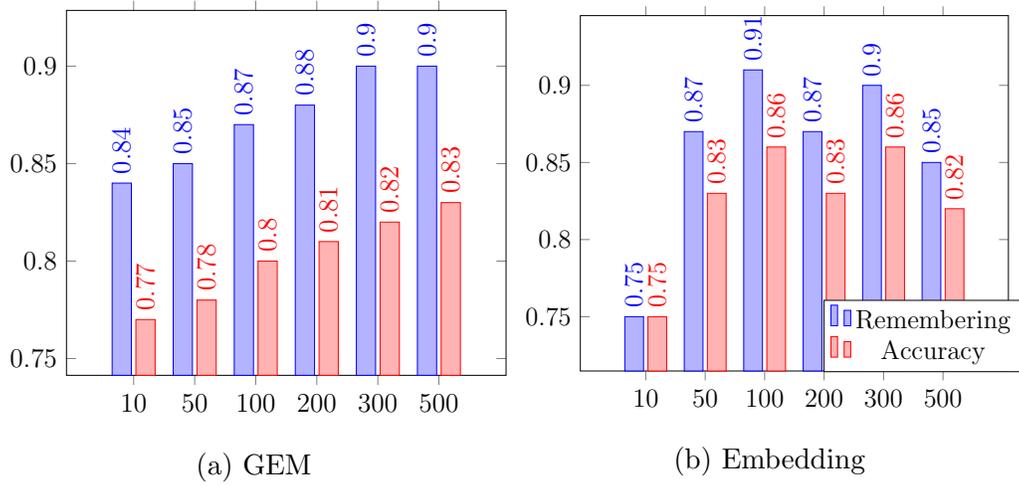


Figure 4: A comparison between GEM and ER when varying the memory size.

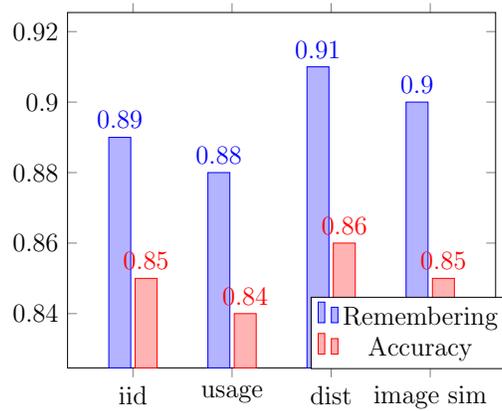


Figure 5: A comparison between the sample size techniques used in ER.

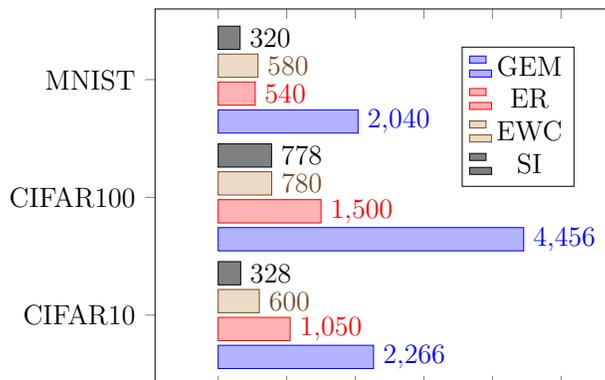
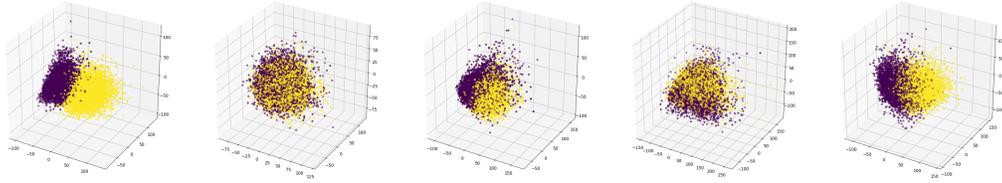


Figure 6: Training time required by the approaches (in seconds). The times are averaged over the ANN experiments.

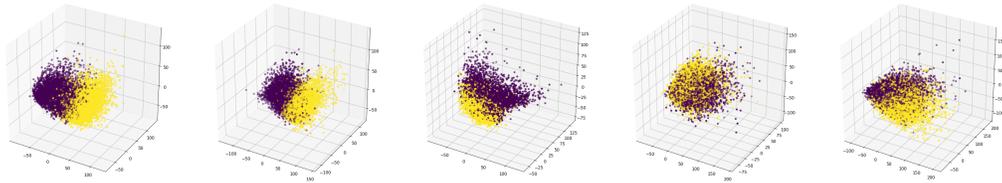
#### 5.4. Visualization of the embeddings

Finally, it could be interesting to see how the features space changes during the training, to see if the intuitions behind the ER method are empirically confirmed. Figure 7 shows that GEM is capable of keeping the embedding shape unchanged during the training, however the division between the classes become less clear in the end of the training. ER method rotates the points, since the distance used are the cosine dissimilarity and it constrains only the direction of the embeddings, but the division remains neat during all the training.

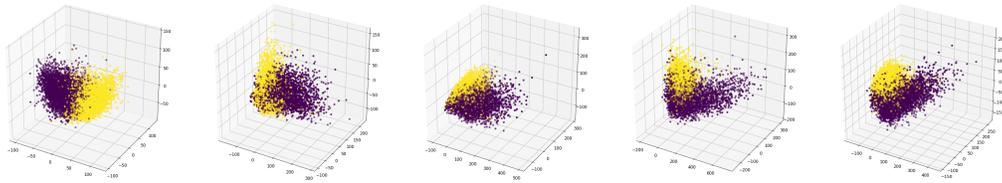
We can now confirm that regularizing the embeddings is a promising approach to alleviate catastrophic forgetting in a CL multi-task scenario.



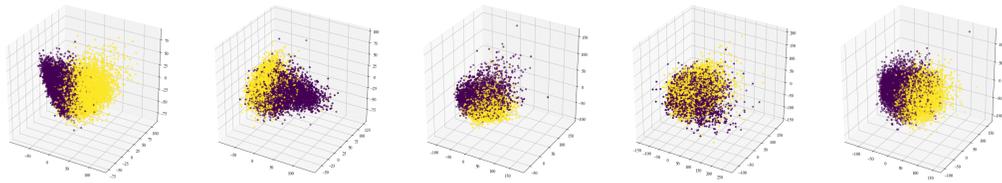
(a) No approach to alleviate catastrophic forgetting



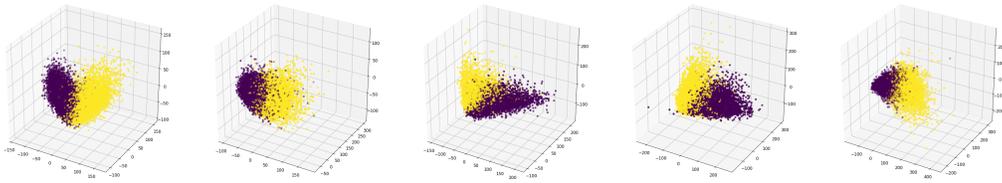
(b) EWC



(c) GEM



(d) SI



(e) ER

Figure 7: How the feature space of the first CIFAR10 task changes from one task to another. Each column contains the feature space at the end of the training of a task from CIFAR10, from 1 to 5. Visualized using PCA algorithm.

## 6. Conclusions

In this paper, we proposed a new method for continual learning of neural networks called embedding regularization, which uses the features vectors extracted from the past tasks to regularize the training on the current task. Embedding has several advantageous characteristics, if compared to other state-of-the-art techniques such as EWC and GEM. First, the method is easier to implement, since it does not work directly with the parameters but let the network adapts itself based on the embeddings; the only constraint is that the network must be able to extract the information from the past tasks. Second, as shown by our experimental evaluation, the method requires very small memory footprint to works properly. The combination of the previous points leads also to a faster training and a better averaged accuracy.

Our method suggests several interesting lines of further research, in which it is not necessary to operate directly on the network to alleviate catastrophic forgetting, but one can design intermediate auxiliary objectives describing the right constraints, such that the network can adapt itself during the training. An interesting line of research is trying to combine the strengths of the proposed ER method with the other state-of-the-art techniques known for CL, also beyond regularization approaches.

## References

## References

- [1] Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P., 2014. Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830.

- [2] Atkinson, C., McCane, B., Szymanski, L., Robins, A., 2018. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. arXiv preprint arXiv:1802.03875.
- [3] Bottou, L., Curtis, F. E., Nocedal, J., 2018. Optimization methods for large-scale machine learning. *SIAM Review* 60 (2), 223–311.
- [4] Bui, T. D., Ravi, S., Ramavajjala, V., 2017. Neural graph machines: Learning neural networks using graphs. arXiv preprint arXiv:1703.04818.
- [5] Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M., 2018. Efficient lifelong learning with A-GEM. CoRR abs/1812.00420.  
URL <http://arxiv.org/abs/1812.00420>
- [6] Chen, T., Goodfellow, I., Shlens, J., 2015. Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641.
- [7] Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7 (Jan), 1–30.
- [8] French, R. M., 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences* 3 (4), 128 – 135.  
URL <http://www.sciencedirect.com/science/article/pii/S1364661399012942>
- [9] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y., 2013. Maxout networks. arXiv preprint arXiv:1302.4389.

- [10] Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
- [11] Huszár, F., 2018. Note on the quadratic penalties in elastic weight consolidation. Proceedings of the National Academy of Sciences, 201717042.
- [12] Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., Yan, S., 2016. Deep learning with s-shaped rectified linear activation units. In: Thirtieth AAAI Conference on Artificial Intelligence.
- [13] Jung, H., Ju, J., Jung, M., Kim, J., 2018. Less-forgetful learning for domain expansion in deep neural networks. In: Proc. Thirty-Second AAAI Conference on Artificial Intelligence.
- [14] 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 114 (13), 3521–3526.
- [15] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al., 2018. Reply to huszár: The elastic weight consolidation penalty is empirically valid. Proceedings of the National Academy of Sciences 115 (11), E2498–E2498.
- [16] LeCun, Y., Cortes, C., Burges, C. J., 1998. The mnist database of handwritten digits.  
URL <http://yann.lecun.com/exdb/mnist>

- [17] Legg, S., Hutter, M., 2007. Universal intelligence: A definition of machine intelligence. CoRR abs/0712.3329.  
URL <http://arxiv.org/abs/0712.3329>
- [18] Li, Z., Hoiem, D., 2017. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (12), 2935–2947.
- [19] Liu, X., Masana, M., Herranz, L., Van de Weijer, J., Lopez, A. M., Bagdanov, A. D., 2018. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In: *Proc. 2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 2262–2268.
- [20] Lopez-Paz, D., Ranzato, M., 2017. Gradient episodic memory for continual learning. In: *Advances in Neural Information Processing Systems*. pp. 6467–6476.
- [21] Oquab, M., Bottou, L., Laptev, I., Sivic, J., 2014. Learning and transferring mid-level image representations using convolutional neural networks. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 1717–1724.
- [22] Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., Wermter, S., 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113, 54–71.
- [23] Qian, S., Liu, H., Liu, C., Wu, S., San Wong, H., 2018. Adaptive activation functions in convolutional neural networks. *Neurocomputing* 272, 204–212.

- [24] Robins, A., 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science* 7 (2), 123–146.
- [25] Rodríguez, N. D., Lomonaco, V., Filliat, D., Maltoni, D., 2018. Don't forget, there is more than forgetting: new metrics for continual learning. CoRR abs/1810.13166.  
URL <http://arxiv.org/abs/1810.13166>
- [26] 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.
- [27] Scardapane, S., Vaerenbergh, S. V., Totaro, S., Uncini, A., 2019. Kafnets: Kernel-based non-parametric activation functions for neural networks. *Neural Networks* 110, 19 – 32.  
URL <http://www.sciencedirect.com/science/article/pii/S0893608018303174>
- [28] Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., Hadsell, R., 2018. Progress & compress: A scalable framework for continual learning. Proc. 34th International Conference on Machine Learning (ICML).
- [29] 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.
- [30] Ulyanov, D., Vedaldi, A., Lempitsky, V. S., 2017. Deep image prior.

CoRR abs/1711.10925.

URL <http://arxiv.org/abs/1711.10925>

[31] Zenke, F., Poole, B., Ganguli, S., 2017. Continual learning through synaptic intelligence. In: Proc. 34th International Conference on Machine Learning (ICML). pp. 3987–3995.

[32] Zenke, F., Poole, B., Ganguli, S., 2017. Improved multitask learning through synaptic intelligence. CoRR abs/1703.04200.

URL <http://arxiv.org/abs/1703.04200>