Memory Aware Synapses
Learning what (not) to forget

Rahaf Aljundi[1], Francesca Babiloni[1], Mohamed Elhoseiny[2],
Marcus Rohrbach[2], and Tinne Tuytelaars[1]

[1] KU Leuven, ESAT-PSI, IMEC, Belgium
[2] Facebook AI Research

Abstract. Humans can learn in a continuous manner. Old rarely utilized knowledge can be overwritten by new incoming information while important, frequently used knowledge is prevented from being erased. In artificial learning systems, lifelong learning so far has focused mainly on accumulating knowledge over tasks and overcoming catastrophic forgetting. In this paper, we argue that, given the limited model capacity and the unlimited new information to be learned, knowledge has to be preserved or erased selectively. Inspired by neuroplasticity, we propose a novel approach for lifelong learning, coined Memory Aware Synapses (MAS). It computes the importance of the parameters of a neural network in an unsupervised and online manner. Given a new sample which is fed to the network, MAS accumulates an importance measure for each parameter of the network, based on how sensitive the predicted output function is to a change in this parameter. When learning a new task, changes to important parameters can then be penalized, effectively preventing important knowledge related to previous tasks from being overwritten. Further, we show an interesting connection between a local version of our method and Hebb's rule, which is a model for the learning process in the brain. We test our method on a sequence of object recognition tasks and on the challenging problem of learning an embedding for predicting <subject, predicate, object> triplets. We show state-of-the-art performance and, for the first time, the ability to adapt the importance of the parameters based on unlabeled data towards what the network needs (not) to forget, which may vary depending on test conditions.

Keywords: Lifelong learning, Hebbian learning, Incremental learning, Continual learning, Adaptation Marcus: I would remove keywords to have space for other things :)

1 Introduction

The (real and digital) world around us evolves continuously. Each day millions of images with new tags appear on social media. Every minute hundreds of hours of video are uploaded on Youtube. This new content contains new topics and trends that may be very different from what one has seen before - think e.g. of new emerging news topics, fashion trends, social media hypes or technical evolutions. Consequently, to keep up to speed, our learning systems should be able to evolve as well.

Fig. 1: Our continuous learning setup. As common in the LLL literature, tasks are learned in sequence, one after the other. If, in between learning tasks, the agent is active and performs the learned tasks, we can use these unlabeled samples to update importance weights for the model parameters. Data that appears frequently, will have a bigger contribution. This way, the agent learns what is important and should not be forgotten.

Yet the dominating paradigm to date, using supervised learning, ignores this issue. It learns a given task using an existing set of training examples. Once the training is finished, the trained model is frozen and deployed. From then on, new incoming data is processed without any further adaptation or customization of the model. Soon, the model becomes outdated. In that case, the training process has to be repeated, using both the previous and new data, and with an extended set of category labels. In a world like ours, such a practice becomes intractable when moving to real scenarios such as those mentioned earlier, where the data is streaming, might be disappearing after a given period of time or even can't be stored at all due to storage constraints or privacy issues.

In this setting, lifelong learning (LLL) [23, 34, 36] comes as a natural solution. LLL studies continual learning across tasks and data, tackling one task at a time, without storing data from previous tasks. The goal is to accumulate knowledge across tasks (typically via model sharing), resulting in a single model that performs well on all the learned tasks. The question then is how to overcome catastrophic forgetting [8, 9, 19] of the old knowledge when starting a new learning process using the same model.

So far, LLL methods have mostly (albeit not exclusively) been applied to relatively short sequences – often consisting of no more than two tasks (e.g. [15, 16, 27]), and using relatively large networks with plenty of capacity (e.g. [1, 6, 32]). However, in a true LLL setting with a never ending list of tasks, the capacity of the model sooner or later reaches its limits and compromises need to be made. Instead of aiming for no forgetting at all, figuring out what can possibly be forgotten becomes at least as important. In particular, exploiting context-specific test conditions may pay off in this case. Consider for instance a surveillance camera. Depending on how or where it is mounted, it always captures images under particular viewing conditions. Knowing how to cope with other conditions is no longer relevant and can be forgotten, freeing capacity for other tasks. This calls for a LLL method that can learn what (not) to forget using unlabeled test data. We illustrate this setup in Figure 1.

Such adaptation and memory organization is what we also observe in biological neurosystems. Our ability to preserve what we have learned before is largely dependent on how frequent we make use of it. Skills that we practice often, appear to be unforgettable, unlike those that we have not used for a long time. Remarkably, this flexibility and adaptation occur in the absence of any form of supervision. According to Hebbian theory [10], the process at the basis of this phenomenon is the strengthening of synapses

connecting neurons that fire synchronously, compared to those connecting neurons with unrelated firing behavior.

In this work, we propose a new method for LLL, coined *Memory Aware Synapses*, or `MAS` for short, inspired by the model of Hebbian learning in biological systems. Unlike previous works, *our LLL method can learn what parts of the model are important using unlabelled data*. This allows for adaptation to specific test conditions and continuous updating of importance weights. This is achieved by estimating importance weights for the network parameters without relying on the loss, but by looking at the sensitivity of the output function instead. This way, our method not only avoids the need for labeled data, but importantly it also avoids complications due to the loss being in a local minimum, resulting in gradients being close to zero. This makes our method not only more versatile, but also simpler, more memory-efficient, and, as it turns out, more effective in learning what not to forget, compared to other model-based LLL approaches.

Contributions of this paper are threefold: *First*, we propose a new LLL method *Memory Aware Synapses* (MAS). It estimates importance weights for all the network parameters in an unsupervised and online manner, allowing adaptation to unlabeled data, e.g. in the actual test environment. *Second*, we show how a local variant of MAS is linked to the Hebbian learning scheme. *Third*, we achieve better performance than state-of-the-art, both when using the standard LLL setup and when adapting to specific test conditions, both for object recognition and for predicting <subject, predicate, object> triplets, where an embedding is used instead of a softmax output.

In the following we discuss related work in section 2 and give some background information in section 3. Section 4 describes our method and its connection with Hebbian learning. Experimental results are given in section 5 and section 6 concludes the paper.

2 Related Work

While lifelong learning has been studied since a long time in different domains (e.g. robotics [36] or machine learning [29]) and touches upon the broader fields of meta-learning [7] and learning-to-learn [2], we focus in this section on more recent work in the context of computer vision mainly.

Method	Type	Constant Memory	Problem agnostic	On Pre-trained	Unlabeled data	Adaptive
LwF [16]	data	✓	X	✓	n/a	X
EBLL ?						
Encoder [27]	data	X	X	X	n/a	X
EWC [11]	model	X	✓	✓	X	X
IMM [15]	model	X	✓	✓	X	X
SI [39]	model	✓	✓	X	X	X
MAS (our)	model	✓	✓	✓	✓	✓

Table 1: LLL desired characteristics and the compliance of methods, that treat forgetting without storing the data, to these characteristics.

The main challenge in LLL is to adapt the learned model continually to new tasks, be it from a similar or a different environment [24]. However, looking at existing LLL solutions, we observe that none of them satisfies all the characteristics one would expect or desire from a lifelong learning approach (see Table 1). First, its memory should be constant w.r.t. the number of tasks, to avoid a gradual increase in memory consumption over time. Second, it should not be limited to a specific setting (e.g. only classification). We refer to this as problem agnostic. Third, given a pretrained model, it should be able to build on top of it and add new tasks. Fourth, being able to learn from unlabeled data would increase the method applicability to cases where original training data no longer

exists. Finally, as argued above, within a fixed capacity network, being able to adapt what not to forget to a specific user setting would leave more free capacity for future tasks. In light of these properties, we discuss recently proposed methods. They can be divided into two main approaches: data-based and model-based approaches. Here, we don't consider LLL methods that require storing samples, such as [17, 28].

Data-based approaches [1, 16, 27, 33] use data from the new task to approximate the performance of the previous tasks. This works best if the data distribution mismatch between tasks is limited. Data based approaches are mainly designed for a classification scenario and overall, the need of these approaches to have a preprocessing step before each new task, to record the targets for the previous tasks is an additional limitation.

Model-based approaches [6, 11, 15, 39], like our method, focus on the parameters of the network instead of depending on the task data. Most similar to our work are [11, 39]. Like them, we estimate an importance weight for each model parameter and add a regularizer when training a new task that penalizes any changes to important parameters. The difference lies in the way the importance weights are computed. In the *Elastic Weight Consolidation* work [11], this is done based on an approximation of the diagonal of the Fisher information matrix. Yet this requires extra computations and storing such a matrix for each task, resulting in memory requirements that grow linearly with the number of tasks. In the *Synaptic Intelligence* work [39], importance weights are computed during training in an online manner. To this end, they record how much the loss would change due to a change in a specific parameter and accumulate this information over the training trajectory. However, also this method has some drawbacks: 1) Relying on the weight changes in a batch gradient descent might overestimate the importance of the weights, as noted by the authors. 2) When starting from a pretrained network, as in most practical computer vision applications, some weights might be used without much changes. As a result, their importance will be underestimated. 3) The computation of the importance is done during training and fixed later. In contrast, we believe the importance of the weights should be able to adapt to the test data where the system is actually applied to. In contrast to the above two methods, we propose to look at the sensitivity of the learned function, rather than the loss. This simplifies the setup considerably since, unlike the loss, the learned function is not in a local minimum, so complications with gradients being close to zero are avoided.

In this work, we propose a model-based method that computes the importance of the network parameters not only in an online manner but also adaptive to the data that the network is tested on in an unsupervised manner. While previous works [25, 30] adapt the learning system at prediction time in a transductive setting, our goal here is to build a continual system that is able to adapt the importance of the weights to what the system actually needs to remember. Our method requires a constant amount of memory and enjoys the main desired characteristics of lifelong learning we listed above while achieving state-of-the-art performance.

3 Background

Standard LLL setup Before introducing our method, we briefly remind the reader of the standard LLL setup, as used e.g. in [1, 15, 16, 27, 39]. It focuses on image classifi-

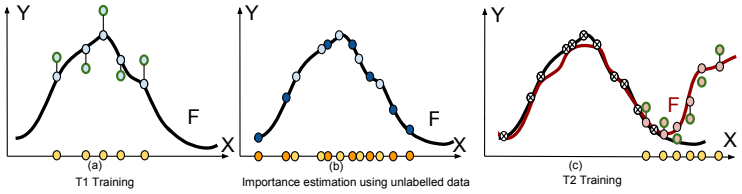


Fig. 2: [39, 11] estimate the parameters importance based on the loss, comparing the network output (light blue) with the ground truth labels (green) using training data (in yellow) (a). In contrast, we estimate the parameters importance, after convergence, based on the sensitivity of the learned function to their changes (b). This allows to use additional unlabeled data points (in orange). When learning a new task, changes to important parameters are penalized, ensuring the function is preserved over the domain densely sampled in (b), while adjusting non important parameters to ensure good performance on the new task (c).

cation and consists of a sequence of disjoint *tasks*, that are learned one after the other. Tasks may correspond to different datasets, or different splits of a dataset, without overlap in category labels. Crucial to this setup is that, when training a task, only the data related to that task is accessible. Ideally, newer tasks can benefit from the representations learned by older tasks (forward transfer). Yet in practice, the biggest challenge is to avoid catastrophic forgetting of the old tasks' knowledge (i.e., forgetting how to perform the old tasks well). This is a far more challenging setup than joint learning, as typically used in the multitask learning literature, where all tasks are trained simultaneously.

Notations We train a single, shared neural network over a sequence of tasks. The parameters $\{\theta_{ij}\}$ of the model are the weights of the connections between pairs of neurons n_i and n_j in two consecutive layers[3]. As in other model-based approaches, our goal is then to compute an importance value Ω_{ij} for each parameter θ_{ij}, indicating its importance with respect to the previous tasks. In a learning sequence, we receive a sequence of tasks $\{T_n\}$ to be learned, each with its training data $(X_n, \hat{Y}_n)$, with X_n the input data and $\hat{Y}_n$ the corresponding ground truth output data (labels). Each task comes with a task-specific loss L_n, that will be combined with an extra loss term to avoid forgetting. When the training procedure converges to a local minimum, the model has learned an approximation F of the true function $\bar{F}$. F maps a new input X to the outputs $Y_1, ..., Y_n$ for tasks $T_1 ... T_n$ learned so far.

4 Our Approach

In the following, we introduce our approach. Like other model-based approaches [11, 39], we estimate an importance weight for each parameter in the network. Yet in our case, these importance weights approximate the *sensitivity of the learned function* to a parameter change rather than a measure of the (inverse of) parameter uncertainty, as in [11], or the sensitivity of the loss to a parameter change, as in [39] (see Figure 2).

[3] In convolutional layers, parameters are shared by multiple pairs of neurons. For sake of clarity, yet without loss of generality, we focus here on fully connected layers.

As it does not depend on the ground truth labels, our approach allows to compute the importance using any available data (unlabeled) which in turn allows for an adaptation to user specific settings. In a learning sequence, we start with task T_1, training the model to minimize the task loss L_1 on the training data $(X_1, \hat{Y_1})$ – or simply using a pretrained model for that task.

4.1 Estimating parameter importance

After convergence, the model has learned an approximation F of the true function $\bar{F}$. F maps the input X_1 to the output Y_1. This mapping F is the target we want to preserve while learning additional tasks. To this end, we measure how sensitive the function F output is to changes in the network parameters. For a given data point x_k, the output of the network is $F(x_k; \theta)$. A small perturbation $\delta = \{\delta_{ij}\}$ in the parameters $\theta = \{\theta_{ij}\}$ results in a change in the function output that can be approximated by:

$$F(x_k; \theta + \delta) - F(x_k; \theta) \approx \sum_{i,j} g_{ij}(x_k)\delta_{ij} \tag{1}$$

where $g_{ij}(x_k) = \frac{\partial(F(x_k;\theta))}{\partial\theta_{ij}}$ is the gradient of the learned function with respect to the parameter θ_{ij} evaluated at the data point x_k and δ_{ij} is the change in parameter θ_{ij}. Our goal is to preserve the prediction of the network (the learned function) at each observed data point and prevent changes to parameters that are crucial for this prediction.

Based on equation 1, we can measure the importance of a parameter by the magnitude of the gradient g_{ij}, i.e. how much does a small perturbation to that parameter change the output of the learned function for data point x_k. We then accumulate the gradients over the given data points to obtain importance weight Ω_{ij} for parameter θ_{ij}:

$$\Omega_{ij} = \frac{1}{N}\sum_{k=1}^{N} || \, g_{ij}(x_k) \, || \tag{2}$$

This equation can be updated in an online fashion whenever a new data point is fed to the network. N is the total number of data points at a given phase. Parameters with small importance weights do not affect the output much, and can therefore be changed to minimize the loss for subsequent tasks, while parameters with large weights should ideally be left unchanged.

When the output function F is multi-dimensional, as is the case for most neural networks, equation 2 involves computing the gradients for each output, which requires as many backward passes as the dimensionality of the output. As a more efficient alternative, we propose to use the gradients of the squared ℓ_2 norm of the learned function output[4], i.e. $g_{ij}(x_k) = \frac{\partial[\ell_2^2(F(x_k;\theta))]}{\partial\theta_{ij}}$. The importance of the parameters is then measured by the sensitivity of the squared ℓ_2 norm of the function output to their changes. This way, we get one scalar value for each sample instead of a vector output. Hence,

[4] We square the ℓ_2 norm as it simplifies the math and the link with the Hebbian method, see section 4.3.

we only need to compute one backward pass and can use the resulting gradients for estimating the parameters importance.

Using our method, for regions in the input space that are sampled densely, the function will be preserved and catastrophic forgetting is avoided. However, parameters not affecting those regions will be given low importance weights, and can be used to optimize the function for other tasks, affecting the function over other regions of the input space.

4.2 Learning a new task

When a new task T_n needs to be learned, we have in addition to the new task loss $L_n(\theta)$, a regularizer that penalizes changes to parameters that are deemed important for previous tasks:

$$L(\theta) = L_n(\theta) + \frac{\lambda}{2} \sum_{i,j} \Omega_{ij} (\theta_{ij} - \theta^*_{ij})^2 \tag{3}$$

with λ a hyperparameter for the regularizer and θ^*_{ij} the "old" network parameters (as determined by the optimization for the previous task in the sequence, T_{n-1}). As such we allow the new task to change parameters that are not important for the previous task (low Ω_{ij}). The important parameters (high Ω_{ij}) can also be reused, via model sharing, but with a penalty when changing them.

Finally, the importance matrix Ω is to be updated after training a new task, by accumulating over the previously computed Ω. Since we don't use the loss function, Ω can be computed on any available data considered most representative for test conditions, be it on the last training epoch, during the validation phase or at test time. In the experimental section 5, we show how this allows our method to adapt and specialize to any set, be it from the training or from the test.

4.3 Connection to Hebbian learning

In this section, we propose a local version of our method, by applying it to a single layer of the network rather than to the network as a whole. Next, we show an interesting connection between this local version and Hebbian learning [10].

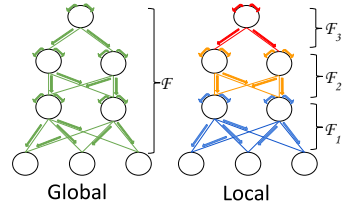


Fig. 3: Gradients flow for computing the importance weight. Local considers the gradients of each layer independently.

A local version of our method Instead of considering the function F that is learned by the network as a whole, we decompose it in a sequence of functions F_l each corresponding to one layer of the network, i.e. $F(x) = F_L(F_{L-1}(...(F_1(x))))$, with L the total number of layers. By locally preserving the output of each layer given its input, we can preserve the global function F. This is further illustrated in Figure 3. Note how "local" and "global" in this context relate to the number of layers over which the gradients are computed.

We use y^k_i to denote the activation of neuron n_i for a given input x_k. Analogous to the procedure followed previously, we consider the squared ℓ_2 norm of each layer after

the activation function. An infinitesimal change $\delta_l = \{\delta_{ij}\}$ in the parameters $\theta_l = \{\theta_{ij}\}$ of layer l results in a change to the squared ℓ_2 norm of the local function F_l for a given input to that layer $y^k = \{y_i^k\} = F_{l-1}(...(F_1(x_k)))$ given by:

$$\ell_2^2(F_l(y^k; \theta_l + \delta_l)) - \ell_2^2(F_l(y^k; \theta_l)) \approx \sum_{i,j} g_{ij}(x_k)\delta_{ij} \tag{4}$$

where $g_{ij}(x_k) = \frac{\partial[\ell_2^2(F_l(y^k;\theta_l))]}{\partial\theta_{ij}}$. In the case of a ReLU activation function, it can be shown that (see supplemental material):

$$g_{ij}(x_k) = 2 * y_i^k * y_j^k \tag{5}$$

Again we consider the accumulation of the gradients evaluated at different data points $\{x_k\}$ as a measure for the importance of the parameter θ_{ij}:

$$\Omega_{ij} = \frac{1}{N}\sum_{k=1}^{N} g_{ij}(x_k) = 2 * \frac{1}{N}\sum_{k=1}^{N} y_i^k * y_j^k \tag{6}$$

Link with Hebbian theory In neuroscience, Hebbian learning theory [10] provides an explanation for the phenomenon of synaptic plasticity. It postulates that "cells that fire together, wire together": the synapses (connections) between neurons that fire synchronously for a given input are strengthened over time to maintain and possibly improve the corresponding outputs. Here we reconsider this theory from the perspective of an artificial neural network after it has been trained successfully with backpropagation. Following Hebb's rule, parameters connecting neurons that often fire together (high activations for both, i.e. highly correlated outputs) are more important for the given task than those that fire asynchronously or with low activations. As such, the importance weight Ω_{ij} for the parameter θ_{ij} can be measured purely locally in terms of the correlation between the neurons' activations, i.e.

$$\Omega_{ij} = \frac{1}{N}\sum_{k=1}^{N} y_i^k * y_j^k \tag{7}$$

The similarity with equation 6 is striking. We can conclude that applying Hebb's rule to measure the importance of the parameters in a neural network can be seen as a local variant of our method that considers only one layer at a time instead of the global function learned by the network. Since only the relative importance weights really matter, the scale factor 2 can be ignored.

4.4 Discussion

Both our global and local method have the advantage of computing the importance of the parameters on any given data point without the need to access the labels or the condition of being computed while training the model. The global version needs to compute the gradients of the output function while the local variant (Hebbian based) can

be computed locally by multiplying the input with the output of the connecting neurons. *Our proposed method (both the local and global version) resembles an implicit memory included for each parameter of the network. We therefore refer to it as Memory Aware Synapses.* It keeps updating its value based on the activations of the network when applied to new data points. It can adapt and specialize to a given subset of data points rather than preserving every functionality in the network. Further, the method can be added after the network is trained. It can be applied on top of any pretrained network and compute the importance on any set of data without the need to have the labels. This is an important criterion that differentiates our work from methods that rely on the loss function to compute the importance of the parameters.

5 Experiments

We start by comparing our method to different existing LLL methods in the standard sequential learning setup of object recognition tasks. We further analyze the behavior and some design choices of our method. Next, we move to the more challenging problem of continual learning of <subject, predicate, object> triplets in an embedding space (section 5.2).

5.1 Object Recognition

We follow the standard setup commonly used in computer vision to evaluate LLL methods [1, 16, 37]. It consists of a sequence of supervised classification tasks each from a particular dataset. Note that this supposes having different classification layers for each task (different "heads") that remain unshared. Moreover, an oracle is used at test time to decide on the task (i.e. which classification layer to use).

Compared Methods - *Finetuning* (`FineTune`). After learning the first task and when receiving a new task to learn, the parameters of the network are finetuned on the new task data. This baseline is expected to suffer from forgetting the old tasks while being advantageous for the new task.

- *Learning without Forgetting [16]* (`LwF`). Given a new task data, the method records the probabilities obtained from the previous tasks heads and uses them as targets when learning a new task in a surrogate loss function. To further control the forgetting, the method relies on first training the new task head while freezing the shared parameters as a warmup phase and then training all the parameters until convergence.

- *Incremental Moment Matching [15]* (`IMM`). A new task is learned with an L2 penalty equally applied to the changes to the shared parameters. At the end of the sequence, the obtained models are merged through a first or second moment matching. In our experiments, mean `IMM` gives better results on the two tasks experiments while mode `IMM` wins on the longer sequence. Thus, we report the best alternative in each experiment.

- *Synaptic Intelligence [39]* (`SI`). This method shows state-of-the-art performance and comes closest to our approach. It estimates the importance weights in an online manner while training for a new task. Similar to our method changes to parameters important for previous tasks are penalized during training of later tasks.

Method	Birds → Scenes		Scenes → Birds		Flower → Birds		Flower → Scenes	
FineTune	45.20 (-8.0)	**57.8**	49.7 (-9.3)	**52.8**	64.87 (-13.2)	**53.8**	70.17 (-7.9)	57.31
LwF [16]	51.65 (-2.0)	55.59	55.89 (-3.1)	49.46	73.97 (-4.1)	53.64	76.20 (-1.2)	58.05
EBLL [37]	52.79 (-0.8)	55.67	56.34 (-2.7)	49.41	75.45 (-2.6)	50.51	76.20 (-1.2)	**58.35**
IMM [15]	51.15 (-2.1)	52.62	54.76 (-4.2)	52.20	75.68 (-2.4)	48.32	76.28 (-1.8)	55.64
EWC [11]	52.19 (-1.4)	55.74	**58.28 (-0.8)**	49.65	76.46 (-1.6)	50.7	77.0 (-1.1)	57.53
SI [39]	52.64 (-1.0)	55.89	57.46 (-1.5)	49.70	75.19 (-2.9)	51.20	76.61 (-1.5)	57.53
MAS (ours)	**53.24 (-0.4)**	55.0	57.61 (-1.4)	49.62	**77.33 (-0.7)**	50.39	**77.24 (-0.8)**	57.38
MAS-GaB	**53.40 (-0.2)**	**63.13**	**58.64 (-0.4)**	50.34	77.16 (-0.9)	51.49	**77.41 (-0.7)**	**59.10**

Table 2: Classification accuracy (%), drop in first task (%)for various sequences of 2 tasks using the object recognition setup.

- *Memory Aware Synapses* (MAS). Unless stated otherwise, we use the global version of our method and with the importance weights estimated only on training data. We use a regularization parameter λ of 1; note that no tuning of λ was performed as we assume no access to previous task data.

Experimental setup We use the AlexNet [14] architecture pretrained on Imagenet [31] from [13][5]. All the training of the different tasks have been done with stochastic gradient descent for 100 epochs and a batch size of 200 using the same learning rate as in [1]. Performance is measured in terms of classification accuracy.

Two tasks experiments We first consider sequences of two tasks based on three datasets: MIT *Scenes* [26] for indoor scene classification (5,360 samples), Caltech-UCSD *Birds* [38] for fine-grained bird classification (5,994 samples), and Oxford *Flowers* [22] for fine-grained flower classification (2,040 samples). We consider: Scene → Birds, Birds→ Scenes, Flower→Scenes and Flower→ Birds, as used previously in [1, 16, 37]. We didn't consider Imagenet as a task in the sequence as this would require retraining the network from scratch to get the importance weights for SI. As shown in Table 2, FineTune clearly suffers from catastrophic forgetting with a drop in performance from 8% to 13%. All the considered methods manage to reduce the forgetting over fine-tuning significantly while having performance close to fine-tuning on the new task. On average, our method followed by SI has the least forgetting (around 1%) while performance on the new task is almost similar ($0-3\%$ lower).

Local vs. global MAS on training/test data Next we analyze the performance of our method when preserving the global function learned by the network after each task (MAS) and its local Hebbian-inspired variant described in section 4.3 (l-MAS). We also evaluate our methods, MAS and l-MAS, when using unlabeled test data and/or labeled training data. Table 3 shows, independent from the set used for computing the importance of the weights, for both l-MAS and MAS the preservation of the previous task and the performance on the current task are quite similar. This illustrates our method ability to estimate the parameters importance of a given task given any set of points, without the need of labeled data. Further, computing the gradients locally at each layer for l-MAS allows for faster computations but less accurate estimations. As such, l-MAS shows an average forgetting of 3% compared to 1% by MAS.

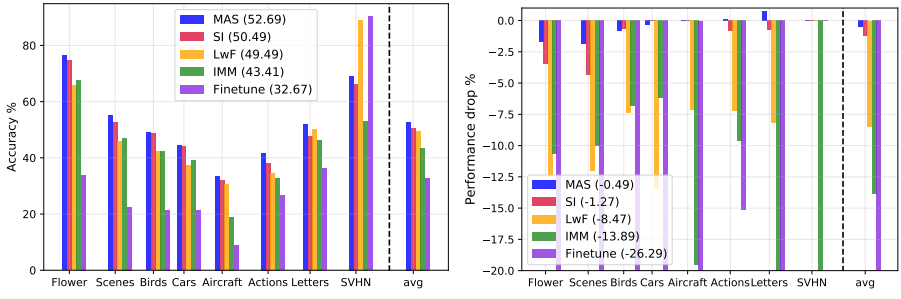
[5] We use the pretrained model available in Pytorch. Note that it differs slightly from other implementations used e.g. in [16].

Method	Ω_{ij} computed on	Birds → Scenes		Scenes → Birds		Flower → Bird		Flower → Scenes	
MAS	Train	53.24	55.0	57.61	49.62	77.33	50.39	77.24	57.38
MAS	Test	53.43	55.07	57.31	49.01	77.62	50.29	77.45	57.45
MAS	Train and Test	53.29	56.04	57.83	49.56	77.52	49.70	77.54	57.39
l-MAS	Train	51.36	55.67	57.61	49.86	73.96	50.5	76.20	56.68
l-MAS	Test	51.62	53.95	55.74	50.43	74.48	50.32	76.56	57.83
l-MAS	Train and Test	52.15	54.40	56.79	48.92	73.73	50.5	76.41	57.91

Table 3: Classification accuracies (%) for the object recognition setup - comparison between using Train and Test data (unlabeled) to compute the parameter importance Ω_{ij}.

ℓ_2^2 vs. vector output We explained in section 4 that considering the gradients of the learned function to estimate the parameters importance would require as many backward passes as the length of the output vector. To avoid this complexity, we suggest to use the square of the ℓ_2 norm of the function in order to get a scalar output. We run two experiments, Flower→Scenes and Flower→ Birds once with computing the gradients with respect to the vector output and once with respect to the ℓ_2^2 norm. We observe no significant difference on forgetting over 3 random trials where we get a mean, over 6 numbers, of $(0.51\% \pm 0.19)$ for the drop on the first task in the vector output case compared to $(0.47\% \pm 0.12)$ for the ℓ_2^2 norm case. No significant difference is observed on the second task either. As such, using ℓ_2^2 is n times faster (where n is the length of the output vector) without loss in performance.

Longer Sequence



(a) (b)

Fig. 4: 4a performance on each task, in accuracy, at the end of 8 tasks object recognition sequence. 4b drop in each task relative to the performance achieved after training each task.

While the two tasks setup gives a detailed look at the average expected forgetting when learning a new task, it remains easy. Thus, we next consider a sequence of 8 tasks. To do so, we add five more datasets: Stanford `Cars` [12] for fine-grained car classification; FGVC-`Aircraft` [18] for fine-grained aircraft classification; VOC `Actions`, the human action classification subset of the VOC challenge 2012 [5]; `Letters`, the Chars74K dataset [3] for character recognition in natural im-

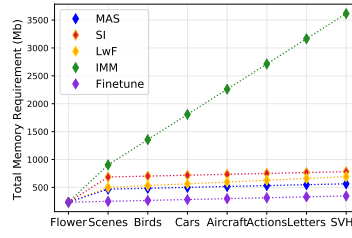


Fig. 5: Overall memory requirement for each method at each step of the sequence.

ages; and the Google Street View House Numbers
SVHN dataset [21] for digit recognition.
Those datasets were also used in [1]. We run the different methods on the following sequence: Flower→Scenes→Birds→Cars→Aircraft→Actions→Letters→SVHN.
While Figure 4a shows the performance on each task at the end of the sequence, 4b shows the observed forgetting on each task at the end of the sequence (relative to the performance right after training that task). The differences between the compared methods become more outspoken. Finetuning suffers from a severe forgetting on the previous tasks while being advantageous for the last task, as expected. LwF [16] suffers from a buildup of errors when facing a long sequence. IMM [15] merges the models at the end of the sequence and the drop in performance differs between tasks. More importantly, the method performance on the last task is highly affected by the moment matching. SI [39] has the least forgetting among our methods competitors. MAS, our method, shows a minimal or no forgetting on the different tasks in the sequence with an average forgetting of 0.49%. It is worth noting that our method's absolute performance on average including the last task is 2% better than SI which indicates our method ability to accurately estimate the importance weights and the new tasks to adjust accordingly. Apart from evaluating forgetting, we analyze the memory requirements of each of the compared methods. Figure 5 illustrates the memory usage of each method at each learning step in the sequence. After Finetune that doesn't treat forgetting, our method has the least amount of memory consumption. Note that IMM grows linearly in storage but at inference time it only uses the obtained model. More details on memory requirements and absolute performances, in numbers, achieved by each method can be found in the supplemental material.

Adaptation Test As we have previously explained, MAS has the ability to adapt the importance weights to a specific subset that has been encountered at test time in an unsupervised and online manner.To test this claim, we have selected one class from the Flower dataset, Krishna Kamal flower. We learn the 8 tasks sequence as above while assuming Krishna Kamal as the only encountered class. Hence, importance weights are computed on that subset only. At the end of the sequence, we observe a minimal forgetting on that subset of 2% compared to 8% forgetting on the Flower dataset as a whole. We also observe higher accuracies on later tasks as only changes to important parameters for that class are penalized, leaving more free capacity for remaining tasks (e.g. accuracy of 84% on the last task, instead of 69% without adaptation). We repeat the experiment with two other classes and obtain similar results. This clearly indicates our method ability to adapt to user specific settings and to learn what (not) to forget.

Single domain / No use of oracle (R1). Here we also test a sequence of 5 permuted MNIST tasks with a 2 layer perceptron (512 units) where the head is shared among the different tasks, i.e. without an oracle. Figure 6 shows the avg. performance and avg. forgetting at the end of the sequence for different values of λ. Alternatively,

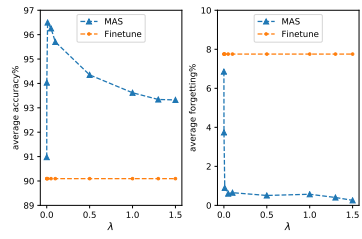


Fig. 6: avg. performance, left, and avg. forgetting, right, on permuted mnist sequence.

methods like Expert Gate [1] could be used at test time to infer which head to use.

Sensitivity of λ in Eqn. 3. Lambda is a tradeoff between the allowed forgetting and the new task loss. We set λ to the largest value that allows an acceptable performance on the new task. For MAS, we used $\lambda = 1$ in all object recognition experiments while for SI and EWC we had to vary λ. Figure 6 shows the effect of λ on the avg. performance and the avg. forgetting in a permuted MNIST sequence. We see the sensitivity around $\lambda = 1$ is very low with low forgetting, although further improvements could be achieved.

5.2 Facts Learning

Next, we move to a more challenging setup where all the layers of the network are shared, including the last layer. Instead of learning a classifier, we learn an embedding space. For this setup, we pick the problem of Fact Learning from natural images [4]. For example, a fact could be "person eating pizza". We design different experimental settings to show the ability of our method to learn what (not) to forget.

Experimental setup We use the 6DS mid scale dataset presented in [4]. It consists of $28,624$ images, divided equally in training and test samples belonging to 186 unique facts. Facts are structured into 3 units: Subject (S), Object (O) and Predicate (P).We use a CNN model based on the VGG-16 architecture [35] pretrained on ImageNet. The last fully connected layer forks in three final layers enabling the model to have three separated and structured outputs for Subject, Predicate and Object as in [4]. The loss minimizes the pairwise distance between the visual and the language embedding. For the language embedding, the Word2vec [20] representation of the fact units is used. To study fact learning from a lifelong perspective, we divided the dataset in tasks belonging to different groups of facts. SGD optimizer is used with a mini-batch of size 35 for 300 epochs and we use a $\lambda = 5$ for our method. For evaluation, we report the fact to image retrieval scenario. We follow the evaluation protocol proposed in [4] and report the mean average precision (MAP). For each task, we consider retrieving the images belonging to facts from this task only. We also report the mean average precision on the whole dataset which differs from the average of the performance achieved on each task. More details can be found in the supplemental materials. We focus on the comparison between the local l-MAS and global MAS variants of our method and SI [39], the best performing method among the different competitors as shown in sec 5.1. Also, to the best of our knowledge, it is the only LLL method that can be easily integrated in such a continual learning scenario.

Two tasks experiments We start by randomly splitting the facts into two groups resulting in two tasks, T_1 and T_2. Our simple sequence is then: $T_1 \rightarrow T_2$. Table 4 shows the performance of each task at the end of the sequence. For the two variants of our method, we show the performance achieved while using training data for computing Ω (i.e. similar setup as in SI). It is clear that this is a much harder task where Finetuning suffers badly from forgetting. The LLL methods manage to control the forgetting but the performance on the second task is lower than with Finetuning. l-MAS achieves 0.264 on

Method	Method evaluated on T_1	T_2	all
FineTune	0.199	**0.57**	0.279
SI [39]	0.279	0.462	0.285
l-MAS	0.264	0.466	0.278
MAS	**0.306**	0.504	**0.290**

Table 4: MAP for the fact learning on the two tasks scenario of a 6DS dataset random split, at the end of the sequence.

Method	Ω_{ij} comp. on	Method evaluated on T_1	T_2	T_3	T_4	all
Finetune	–	0.19	0.19	0.28	**0.71**	0.18
SI[39]	Train	0.36	0.32	0.38	0.68	0.25
MAS	Train	0.42	**0.37**	0.41	0.65	0.29
MAS	Train&Test	**0.43**	**0.37**	**0.46**	0.66	**0.30**

Table 5: MAP for the fact learning on the 4 tasks random split, from the 6DS dataset, at the end of the sequence.

the first task, which is slightly lower than the 0.279 obtained by SI. MAS scores the best on T_1 with 0.306.

Longer sequence of tasks Next, we consider a sequence of 4 tasks obtained of randomly splitting the facts of the same dataset into 4 groups. Table 5 presents the achieved performance on each set of the 4 tasks at the end of the learned sequence. Similar to previous experiments, Finetune is only advantageous on the last task while drastically suffering on the previous tasks. However, here, our method differentiates itself clearly, showing 6% better MAP on the first two tasks compared to SI. Overall, MAS achieves a MAP of 0.29 compared to 0.25 by SI and only 0.18 by Finetune. When MAS importance weights are computed on both training and test data, a further improvement is achieved with 0.30 overall performance. This highlights our method ability to benefit from extra unlabeled data to further enhance the importance estimation.

Adaptation Test

Finally we want to test the ability of our method in learning not to forget a specific subset of a task. When learning a new task, we care about the performance on that specific set more than the rest. For that reason, we clustered the dataset into 4 disjoint groups of facts, representing 4 tasks, and then selected a specialized subset of T_1, namely 7 facts of person playing sports. More details on the split can be found in the supplemental material. We run our method with the importance parameters computed only over the examples from this set along the 4 tasks sequence. Figure 7 shows the achieved performance on this sport subset by each method at each step of the learning sequence.

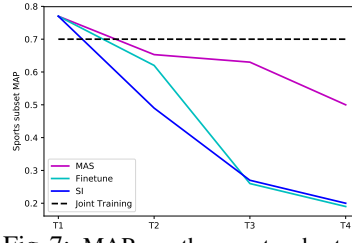


Fig. 7: MAP on the sport subset of the 6DS dataset after each task in a 4 tasks sequence. MAS managed to learn that the sport subset is important to preserve and prevents significantly the forgetting on this subset.

Joint Training (black dashed) is shown as reference. It violates the LLL setting as it trains on all data jointly. Note that SI can only learn importance weights during training, and therefore cannot adapt to a particular subset. Our MAS (pink) succeeds to learn that this set is important to preserve and achieves a performance of 0.50 at the end of the sequence, while the performance of finetuning and SI on this set was close to 0.20.

6 Conclusion

In this paper we argued that, given a limited model capacity and unlimited evolving tasks, it is not possible to preserve all the previous knowledge. Instead, agents should learn what (not) to forget. Forgetting should relate to the rate in which a specific piece of knowledge is used. This is similar to how biological systems are learning. In the absence of error signals, synapses connecting biological neurons strengthen or weaken based on the concurrence of the connected neurons activations. In this work and inspired by the synaptic plasticity, we proposed a method that is able to learn the importance of network parameters from the input data that the system is active on, in an unsupervised manner. We showed that a local variant of our method can be seen as an application of Hebb's rule in learning the importance of parameters. We first tested our method on a sequence of object recognition problems in a traditional LLL setting. We then moved to a more challenging test case where we learn facts from images in a continuous manner. We showed i) the ability of our method to better learn the importance of the parameters using training data, test data or both; ii) state-of-the-art performance on all the designed experiments and iii) the ability of our method to adapt the importance of the parameters towards a frequent set of data. We believe that this is a step forward in developing systems that can always learn and adapt in a flexible manner.

References

1. Aljundi, R., Chakravarty, P., Tuytelaars, T.: Expert gate: Lifelong learning with a network of experts. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
2. Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M.W., Pfau, D., Schaul, T., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 3981–3989. Curran Associates, Inc. (2016)
3. de Campos, T.E., Babu, B.R., Varma, M.: Character recognition in natural images. In: Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal (February 2009)
4. Elhoseiny, M., Cohen, S., Chang, W., Price, B.L., Elgammal, A.M.: Sherlock: Scalable fact learning in images. In: AAAI. pp. 4016–4024 (2017)
5. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html
6. Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D.: Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint arXiv:1701.08734 (2017)
7. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the International Conference on Machine Learning (ICML) (2017)
8. French, R.M.: Catastrophic forgetting in connectionist networks. Trends in cognitive sciences **3**(4), 128–135 (1999)
9. Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A., Bengio, Y.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211 (2013)

10. Hebb, D.: The organization of behavior. 1949. New York Wiely (2002)
11. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. arXiv preprint arXiv:1612.00796 (2016)
12. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3d object representations for fine-grained categorization. In: Proceedings of the IEEE International Conference on Computer Vision Workshops. pp. 554–561 (2013)
13. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997 (2014)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012)
15. Lee, S.W., Kim, J.H., Ha, J.W., Zhang, B.T.: Overcoming catastrophic forgetting by incremental moment matching. arXiv preprint arXiv:1703.08475 (2017)
16. Li, Z., Hoiem, D.: Learning without forgetting. In: European Conference on Computer Vision. pp. 614–629. Springer (2016)
17. Lopez-Paz, D., et al.: Gradient episodic memory for continual learning. In: Advances in Neural Information Processing Systems. pp. 6470–6479 (2017)
18. Maji, S., Kannala, J., Rahtu, E., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. Tech. rep. (2013)
19. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of learning and motivation **24**, 109–165 (1989)
20. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
21. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)
22. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing (Dec 2008)
23. Pentina, A., Lampert, C.H.: Lifelong learning with non-i.i.d. tasks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 1540–1548 (2015)
24. Pentina, A., Lampert, C.H.: Lifelong learning with non-iid tasks. In: Advances in Neural Information Processing Systems. pp. 1540–1548 (2015)
25. Quadrianto, N., Petterson, J., Smola, A.J.: Distribution matching for transduction. In: Advances in Neural Information Processing Systems. pp. 1500–1508 (2009)
26. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 413–420. IEEE (2009)
27. Rannen, A., Aljundi, R., Blaschko, M.B., Tuytelaars, T.: Encoder based lifelong learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1320–1328 (2017)
28. Rebuffi, S.A., Kolesnikov, A., Lampert, C.H.: icarl: Incremental classifier and representation learning. arXiv preprint arXiv:1611.07725 (2016)
29. Ring, M.B.: Child: A first step towards continual learning. Machine Learning **28**(1), 77–104 (1997)
30. Royer, A., Lampert, C.H.: Classifier adaptation at prediction time. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1401–1409 (2015)

31. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
32. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016)
33. Shmelkov, K., Schmid, C., Alahari, K.: Incremental learning of object detectors without catastrophic forgetting. In: The IEEE International Conference on Computer Vision (ICCV) (2017)
34. Silver, D.L., Yang, Q., Li, L.: Lifelong machine learning systems: Beyond learning algorithms. In: AAAI Spring Symposium: Lifelong Machine Learning. pp. 49–55. Citeseer (2013)
35. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
36. Thrun, S., Mitchell, T.M.: Lifelong robot learning. Robotics and autonomous systems **15**(1-2), 25–46 (1995)
37. Triki, A.R., Aljundi, R., Blaschko, M.B., Tuytelaars, T.: Encoder based lifelong learning. arXiv preprint arXiv:1704.01920 (2017)
38. Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., Perona, P.: Caltech-UCSD Birds 200. Tech. Rep. CNS-TR-2010-001, California Institute of Technology (2010)
39. Zenke, F., Poole, B., Ganguli, S.: Improved multitask learning through synaptic intelligence. In: Proceedings of the International Conference on Machine Learning (ICML) (2017)