In Defense of the Learning Without Forgetting for Task Incremental Learning

Guy Oren and Lior Wolf
Tel-Aviv University
{guyoren347, liorwolf}@gmail.com

Abstract

Catastrophic forgetting is one of the major challenges on the road for continual learning systems, which are presented with an on-line stream of tasks. The field has attracted considerable interest and a diverse set of methods have been presented for overcoming this challenge. Learning without Forgetting (LwF) is one of the earliest and most frequently cited methods. It has the advantages of not requiring the storage of samples from the previous tasks, of implementation simplicity, and of being well-grounded by relying on knowledge distillation. However, the prevailing view is that while it shows a relatively small amount of forgetting when only two tasks are introduced, it fails to scale to long sequences of tasks. This paper challenges this view, by showing that using the right architecture along with a standard set of augmentations, the results obtained by LwF surpass the latest algorithms for task incremental scenario. This improved performance is demonstrated by an extensive set of experiments over CIFAR-100 and Tiny-ImageNet, where it is also shown that other methods cannot benefit as much from similar improvements. Our code is available at: `https://github.com/guy-oren/In_defence_of_LWF`

1. Introduction

The phenomenon of catastrophic forgetting (CF) of old concepts as new ones are learned in an online manner is well-known. The approaches to overcome it can be categorized, as suggested by De Lange *et al*. [3], into three families: (i) replay-based methods, which store selected samples of previously encountered classes, (ii) regularization-based methods, that limit the freedom to learn new concepts, and (iii) parameter isolation methods, which directly protect the knowledge gained in the past, by dividing the network parameters into separate compartments.

The field of continual learning is very active, with dozens of methods that have emerged in the last few years. However, it seems that the growing interest leads to confusion rather than to the consolidation of knowledge. As practitioners looking to find out which online learning method would be suitable for a real-world application, we were unable to identify the solid methods of the field and could not infer from the literature the guiding principles for tackling catastrophic forgetting.

Indeed, reviewing the literature, one can find many insightful ideas and well-motivated solutions. However, little data regarding the generality of continual learning methods, the sensitivity of the methods to the specific setting and hyperparameters, the tradeoff between memory, run-time and performance, and so on. Ideally, one would like to find a method that is not only well-grounded and motivated, but also displays a set of desired properties: (i) work across multiple datasets, (ii) be stable to long sequences of on-line learning tasks, (iii) benefit from additional capacity, (iv) display flexibility in network architecture that allows the incorporation of modern architectures, (v) display an intuitive behavior when applying regularization, and (vi) present robustness to hyperparameters.

We demonstrate that these properties hold for one of the first methods to be proposed for tackling CF, namely the Learning without Forgetting (LwF) method [22]. This is a bit surprising, since this method, as a classical method in a fast-evolving field, has been repeatedly used as an inferior baseline. However, we show that unlike many of the more recent methods, this scapegoat method can benefit from residual architectures and further benefits from simple augmentation techniques. Moreover, while the original LwF implementation employed techniques such as warmup and weight decay, we were able to train without these techniques and their associated hyperparameters. Overall, we find LwF, which is a simple data-driven regularization technique, to be more effective than the most promising regularization-based and parameter-isolation methods.

2. Related work

It is often the case that new methods are presented as having clear advantages over existing ones, based on empirical evidence. The inventors of these methods have little incentive to explore the underlying reason for the performance gap. Without a dedicated effort to do so, the litera-

ture can quickly become misleading.

In our work, we demonstrate that the task-incremental learning methods that have emerged since the 2016 inception of the LwF method are not more accurate than this straightforward method. This demonstration is based on changing the underlying neural network architecture to a ResNet [10] and on employing a simple augmentation technique during training. Moreover, we show that LwF benefits from more capacity, width wise.

A recent related attempt by De Lange *et al*. [3] also addresses the need to compare multiple continual learning algorithms in task-incremental settings. That study has employed multiple architectures, and, similar to us, have noted that the LwF method benefits from the additional capacity given by extra width but not from extra depth. However, ResNets or augmentations were not employed and the conclusion was that LwF is not competitive with the more recent techniques. This conclusion is in sheer contrast to ours, demonstrating the challenge of comparing methods in a way that exposes their full potential, and the need to perform such comparative work repeatedly.

2.1. Task-incremental learning

CF in neural networks has been observed from the beginning. However, there is no consensus regarding the proper settings and metrics for comparing different techniques. In this work, we adopt a setting definition from the work of [33, 12], who define three different settings for continual learning – task incremental, domain incremental, and class incremental. In all scenarios, the system is presented with a stream of tasks and is required to solve all tasks that are seen so far. In task incremental, the task identifier is provided both in train and inference time. In domain incremental, the task identifier is provided only in train time, and the classifier does not need to infer the task identifier but rather just solve the task at hand. In class incremental, the learner also needs to infer the task identifier in inference time.

We focus on the task incremental setting. Moreover, we do not consider replay-based methods since these rely heavily on accessing data retained from the previous tasks, which is not desirable in real-world scenarios, and depends on an additional parameter that is the size of the memory.

The literature has a great number of methods, further emphasizing the need for comparative work. In this work, we focus on the methods that are repeatedly reported in the literature [3, 29, 13, 21]. These include: Elastic Weight Consolidation (EWC; [16], online version), Incremental Moment Matching (IMM; [20], both Mean and Mode variants), overcoming CF with Hard Attention to the Task (HAT; [29]), continual learning with Hypernetworks (Hyper-CL; [34]) and Adversarial Continual Learning (ACL; [4]).

Both the EWC and IMM variants, belong to a regularization-based family and add a structural, weight-based, regularization term to the loss function to discourage changes to weights that are important for previous tasks. IMM performs a separate model-merging step after learning a new task, which EWC does not. Although this family of methods is very rich, IMM and EWC are among the leading methods and are often cited as baselines.

The HAT approach belongs to the parameter isolation family and applies a light-weight, unit-based, learnable, and 'soft' masks per task. HAT is a successor to various works, including (i) progressive neural networks (PNNs; [27]), which applies a complete and separate network for each task (columns) with adapters between columns, (ii) PathNet [5] that also pre-assigns some amount of network capacity per task but, in contrast to PNNs, avoids network columns and adapters and instead suggests to learn evolutionary the paths between modules, and (iii) PackNet [24], which uses weight-based pruning heuristics and a retraining phase to maintain a binary mask for each task. Since HAT was shown to have both performance and computational advantages over (i)-(iii), we focus on it as a representative method from this line of work.

Hyper-CL [34], a recent addition to the parameter isolation family, belongs to a different branch in this family than HAT. Instead of using a fixed pre-determined capacity, Hyper-CL suggests learning the weights of a target network for each task. Hyper-CL employs a variant of Hypernetworks [8], called Chunked-Hypernetworks [25], which generates different subsets of the target network's parameters using the same generator. To do so, the method learns both the task embedding and the "chunk" embedding. This variant makes it possible to maintain a much smaller hypernetwork than the target network. To overcome CF, they apply regularization that constrains the weights of the previously seen target task from changing.

Some methods belong to more than one category. ACL [4] employs both parameter isolation using a small private network for each task, and regularization for a shared network across tasks. This regularization contains two parts: an adversarial loss that makes the shared encoding task-independent [6] and a disentanglement loss that acts to remove the overlap between the private- and the shared-encoding [28].

Naturally, given the number of relevant methods, it is not feasible to compare with all of them. The regularization-based family presents two additional methods that we considered: Encoder Based Lifelong Learning (EBLL; [26]) and Memory Aware Synapses (MAS; [1]). EBLL extends LwF by adding a per-task auto-encoder, requiring further hyperparameter tuning. The literature shows that it only marginally improves over LwF for AlexNet-like architectures [3, 1], and our attempts to apply it together with ResNets led to poor results. MAS was also shown in [3] to only slightly improved over LwF.

3. The LwF method and its modifications

The LwF method by Li *et al*. [22], belongs to the regularization-based family. However, unlike EWC and IMM, its regularization is data-driven. The method seeks to utilize the knowledge distillation loss [11] between the previous model and the current model to preserve the outputs of the previous task. Since maintaining the data of previous tasks is not desirable and rather not scalable, LwF uses only the current task data for knowledge distillation.

In the task-incremental setting, the learner is given a new set of labels to learn at each round. This set of classes is called a task. In LwF the classifier is composed out of two parts: the feature extractor f and a classifier head c_i per each task for $i = 1, 2, \ldots, T$.

Let $\{(x_j^t, y_j^t)\}$ be the set of training samples for task t. The cross-entropy loss is used as the primary loss for training the classifier $c_t \circ f$:

$$L_{CE} = -\sum_j \log[c_t(f(x_j^t))]_{y_j^t} \tag{1}$$

where the subscript y_j^t is used to denote the pseudo-probability of the classifier for the ground truth label.

When learning a new task t, to maintain previous task knowledge, we employ knowledge distillation between the "old" feature extraction and the previous task classifier heads and the new ones. These are denoted by f^o for the previous feature extractor network (as learned after task $t-1$), and c_i^o for $i = 1, 2, \ldots, t-1$ for the previous heads. The learned feature extraction is denoted by f and the updated task classifiers are denoted by c_i, for $i = 1, 2, \ldots t$.

For simplicity, we described the knowledge distillation process for one previous task and one sample $(x, y) \in \{(x_j^t, y_j^t)\}$ from the current task t. However, the process is repeated for the classifier heads of all previous tasks and all samples of task t, while summing up the individual losses. Let $Y^o := [y_1^o, y_2^o, ...] = c_i^o(f^o(x))$ be the vector of probabilities that the old classifier of task i assigns to sample x. Similarly, let $Y := [y_1, y_2, ...]$ be the vector of probabilities for the same training samples obtained with $c_i \circ f$. To apply the knowledge distillation loss, these vectors are modified in accordance with some temperature parameter θ:

$$y'_k = \frac{y_k^{\frac{1}{\theta}}}{\sum_m y_m^{\frac{1}{\theta}}}, \quad y'^o_k = \frac{(y_k^o)^{\frac{1}{\theta}}}{\sum_m (y_m^o)^{\frac{1}{\theta}}}. \tag{2}$$

The temperature is taken to be larger than one, to increase small probability values and reduce the dominance of the high values. The knowledge distillation loss is defined as:

$$L_{dist} = -\sum_k y'^o_k log(y'_k), \tag{3}$$

where the summation is done over all labels of task i.

We followed the author's suggestions and in all our experiments and set $\theta = 2$ and the regularization weight to one, *i.e*., the knowledge distillation loss had the same weight as the classification loss of the new task. It is worth mentioning that although the original LwF work [22] evaluated the method in the two task scenario, it can be readily extended to any number of tasks by using knowledge distillation loss over all $c_i^o, i = 1, 2 \ldots, t-1$. This further highlights the need for performing our research, since such an extension was previously done in the context of attempting to present the preferable performance of a new method. We also note that it was suggested in [22] to use a warmup phase at the beginning of training for each new task, in which both f and $c_i, i = 1, 2, \ldots, t-1$ are frozen and one trains c_t with the cross-entropy loss until convergence. However, since the effect of this seems negligible even in the original paper, we do not perform this. The authors also used regularization in the form of weight decay during training, which we remove to avoid the need to fit a regularization hyperparameter for each experiment. Moreover, in our initial experiments weight decay tends to hurt the accuracy of new tasks.

3.1. Architecture

Li *et al*. [22] employed AlexNet [18] and VGGNet [30] to evaluate the performance of the method. Interestingly, even the recent review work by De Lange *et al*. [3] uses AlexNet as a reference network, despite ongoing advances in network architectures. There is also a key difference between the different versions of AlexNet-like architectures employed in [22] and [29]. The latter use Dropout [31], which as we show empirically, is detrimental.

We also offer to use the ResNet [10] architecture. We are not the first to attempt to use ResNets for LwF. Mallya *et al*. [24] employed LwF with a ResNet-50 network as an underperforming baseline. However, our experiments demonstrate that LwF mostly benefits from a Wide-ResNet [35] network rather than from deeper ones.

3.2. Data augmentation

Using a method with a shared model presents a challenge. On the one hand, the shared part must have enough capacity to learn new tasks. On the other hand, bigger networks are more vulnerable to overfitting when training on the first tasks. The parameter isolation family works around this problem by dynamically changing the capacity of the network as in PNNs [27] or learning a specific target network for each task with enough capacity for each task, like in Hyper-CL [34].

In addition to the capacity needs, another challenge that the LwF method faces is the need to mitigate the difference between the input distributions for different tasks. In the

extreme, where the input distributions are very dissimilar, the knowledge distillation loss is no longer constraining the network to success on previous tasks.

Data augmentation, which is a well-studied technique for overcoming overfitting by virtually expending the dataset at hand, also has the potential to close the gap between different input distributions and therefore reduce forgetting. In our experiments, we employ a very basic set of augmentation consisting of random horizontal flips, color jitter (randomly change the brightness, contrast, saturation, and hue), and translation. As it turns out, these are sufficient to reduce the forgetting almost to zero, while substantially increasing the average accuracy for all tested settings.

4. Experiments

The common datasets for evaluating CF in classification problems include permutations of the MNIST data [32], a split of the MNIST data [20], incrementally learning classes of the CIFAR data sets [23], or on considering two datasets and learning the transfer between them [22]. Serrà *et al*. [29] points out the limitations of the MNIST setups, since these do not well represent modern classification tasks. The two-task scenario is criticized for being limited and does not enable the evaluation of CF for sequential learning with more than two tasks. CIFAR-100 splits are criticized for having tasks that are relatively similar in nature. However, in our experiments, performance on CIFAR-100 splits discriminates well between different methods and between different settings of the same method.

In addition to CIFAR-100 [17], we employ Tiny-ImageNet [19] in our experiments. The latter presents a higher diversity with more classes and the ability to challenge methods with longer and more meaningful sequences of tasks. To obtain a generic estimate, we shuffle the order of classes in each dataset and repeat each experiment setup five times with different seeds.

A common CIFAR setup, introduced in [36] offers to use CIFAR-10 as a first task, then split CIFAR-100 into five distinct tasks with 10 disjoint classes each. However, it may introduce a bias in evaluating task-incremental methods, since it makes the first task much larger and, therefore, conceals the problem of first-task overfitting. In this work, we consider a different setting, in which CIFAR-100 is divided into 5-Splits (i.e., 5-tasks), 10-Splits, and 20-Splits with 20, 10, and 5 classes in each task, respectively. Each class in CIFAR-100 contains 500 training images and 100 testing images. Each image size is $3 \times 32 \times 32$. As a validation set, we shuffle the training data and use 90% as training examples and 10% as validation examples.

A recent work by De Lange *et al*. [3] employed Tiny-ImageNet as a benchmark using a similar setup to the CIFAR-100 setup above. However, they split the dataset to 20 disjoint tasks with 10 classes each. Since we opt for a longer sequence of tasks while still keeping them meaningful, we split the dataset into 40 disjoint tasks with 5 classes each. As our results will show, this setting pushes the limits of the task-incremental methods.

Each class in Tiny-ImageNet contains 500 training images, 50 validation images, and 50 testing images. The original image size for this dataset is $3 \times 64 \times 64$. Since the test set is not publicly available, we use the validation set as a test set and as a validation set, we shuffle the training data and use 90% for training and 10% for validation.

To evaluate performance, we adopt the metrics of [23]:

$$\textbf{Average Accuracy: } \text{ACC} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i} \tag{4}$$

$$\textbf{Backward Transfer: } \text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i} \tag{5}$$

where T is the number of tasks and $R_{i,j}$ is the test accuracy score for task j after the model learned task i. We note that $BWT < 0$ reports CF, while $BWT > 0$ indicates that learning new tasks helped the preceding tasks.

4.1. The effect of the network architecture

We first present experiments for LwF with various network architectures and no data augmentation. The AlexNet-like architecture [18] we use follows [29] and has three convolutional layers of 64, 128, and 256 filters with 4×4, 3×3, and 2×2 kernel sizes, respectively. On top, there are two fully-connected layers of 2048 units each. This network employs rectified linear units (ReLU) as activations, and 2×2 max-pooling after the convolutional layers. A Dropout of 0.2 is applied for the first two layers and 0.5 for the rest. All layers are randomly initialized with Xavier uniform initialization [7].

While LwF is commonly used with an AlexNet-like architecture [21, 29, 3], we opt to use more modern architectures. We choose to use the popular architecture family of ResNets. In this work, we use ResNet-20 (RN-20), ResNet-32 (RN-32) and ResNet-62 (RN-62) [10], as well as Wide-ResNet-20 networks with width factors 2 or 5 [35] (WRN-20-W2 and WRN-20-W5 respectively). Those networks employ ReLU activations and Batch Normalization layers [14]. All convolutional layers were randomly initialized with Kaiming normal inits with fan-out mode [9], and the normalization layers were initialized as constants with 1 and 0 for weight and bias, respectively. All architecture tested use separated fully-connected layers with a softmax output for each task as a final layer. More details can be found in the supplementary.

In all experiments, LwF is trained up to 200 epochs for each task. We use a batch size of 64 and an SGD optimizer with a learning rate of 0.01 and a momentum of 0.9. We

		CIFAR 5-Split		CIFAR 10-Split		CIFAR 20-Split		Tiny-ImageNet 40-Split	
Arch.	#Params	BWT	ACC	BWT	ACC	BWT	ACC	BWT	ACC
AlexNet-D	6.50	-39.9 ± 1.4	36.6 ± 1.5	-52.9 ± 1.2	28.1 ± 1.3	-54.4 ± 1.1	31.3 ± 0.8	-50.5 ± 1.0	25.0 ± 0.4
AlexNet-ND	6.50	-1.8 ± 0.6	56.6 ± 1.1	-2.9 ± 0.2	67.0 ± 1.0	-3.1 ± 0.3	75.5 ± 0.6	-2.8 ± 0.3	66.9 ± 0.8
RN-20	0.27	-0.4 ± 0.3	60.4 ± 0.7	-1.9 ± 0.5	67.2 ± 1.0	-2.3 ± 0.4	76.2 ± 0.8	-3.0 ± 0.5	70.8 ± 1.0
RN-32	0.47	-1.8 ± 0.7	58.8 ± 2.0	-1.8 ± 0.2	67.1 ± 1.1	-2.7 ± 0.2	75.6 ± 0.4	-2.4 ± 0.2	70.9 ± 1.1
RN-62	0.95	-1.7 ± 0.6	58.9 ± 0.7	-2.7 ± 0.4	66.0 ± 0.8	-2.9 ± 0.4	75.6 ± 0.7	-3.1 ± 0.9	70.3 ± 1.2
WRN-20-W2	1.08	-1.2 ± 0.6	62.0 ± 0.3	-2.1 ± 0.6	69.6 ± 0.8	-3.3 ± 0.4	77.3 ± 0.4	-3.8 ± 0.2	71.5 ± 0.6
WRN-20-W5	6.71	-2.0 ± 0.5	64.2 ± 1.1	-2.9 ± 0.3	71.2 ± 0.5	-3.7 ± 0.3	79.4 ± 0.6	-4.5 ± 0.3	72.6 ± 0.8

Table 1. Network results summary for LwF. BWT and ACC in %. #Params in millions and counts only for the shared feature extractor. All results are averaged over five runs with standard deviations. D=Dropout, ND=No Dropout, RN=ResNet, WRN=WideResNet.

used the validation set to schedule the learning rate, where we drop the learning rate by a factor of 3 if there is no improvement in the validation loss for 5 epochs. Training is stopped when the learning rate becomes lower than 10^{-4}.

The results are depicted in Tab. 1. Our clearest and most significant result is that the underlying network has a great effect on LwF performance. While LwF with AlexNet with Dropout architecture greatly suffers from forgetting which results in low ACC, just removing the Dropout from the network results in a sizable performance boost. This makes sense while using Dropout on the teacher side creates a strong teacher that can be viewed as a large ensemble of models that shares weight [11], on the student side, this weakens the regularization of LwF. Randomly choosing which weights to regularize ignores their importance for older tasks, which results in high forgetting.

Next, switching to RN-20 with an order of magnitude fewer parameters shows preferable performance. This change reveals the potential of LwF to obtain competitive ACC and BWT.

Following [3] we investigate the effect of width and depth of the architecture with the ResNet network on LwF performance. We used two deeper networks (RN-32 and RN-62) and two wider networks (WRN-20-W2 and WRN-20-W5). Our results (Tab. 1) show that while using a deeper network gives similar or inferior results compare to RN-20, using wider networks increases performance.

4.2. The effect of data augmentation

We conjectured in Sec. 3.2 that LwF performance can be further increased by using data augmentations. In this section, we conduct experiments on WRN-20-W5, which is the best performer among the tested architectures, with a relatively simple set of random augmentations: random horizontal translation of up to 3 pixels with reflection padding, random horizontal flip, and color jitter (brightness, contrast and saturation with jitter of 0.3 and hue with jitter of 0.2).

The results are summarized in Tab. 2. As can be observed, applying augmentation in this setting leads to improvement in both ACC and BWT. Therefore, there is no trade-off between accuracy and forgetting. We emphasize that even though no augmentations protocol search was conducted and that the set of augmentations in use is rather small and simple, the performance boost is substantial.

4.3. Comparison with other methods

We consider two regularization-based methods: EWC [16] and IMM [20] and two parameter isolation methods: HAT [29] and Hyper-CL [34]. ACL [4] is considered as a recent hybrid method. As an upper bound for overall performance we consider a joint training method (JOINT), which for each incoming task, trains on the data of all tasks seen so far. The hyper-parameters for EWC, IMM and HAT were the best found in [29] and for Hyper-CL to the best found in [34]. For ACL, we quote the results mentioned in the paper, *i.e*. for AlexNet-like architecture with Dropout (both private and shared) and no augmentations at all.

Following our findings for LwF, we opt to use all baseline methods with WRN-20-W5. However, we found that none of the baseline methods performs well with it. We found that some of the baseline methods are tightly coupled with the architecture originally presented in the paper. The authors of Hyper-CL [34] did an extensive hyperparameter search for both the hypernetwork and target architectures. They conclude that it is crucial to choose the right combination since it has a great effect on performance. Therefore, we used the best Hypernetwork-Target pair they found for the "chunked", more effective, version. This pair consists of a hypernetwork which has a linear layer that maps task and chunk embedding of size 32 each to a chunk of size 7000 of a ResNet-32 target network. Another coupling we found was for the HAT method, we could not achieve reasonable performance with an underlying ResNet architecture. We conjecture that the masking process in HAT needs to be adapted for usage with batch normalization layers, and report results with the AlexNet-like network presented by Serrà *et al*. [29].

Both EWC and IMM, although not coupled with specific architecture, were found to be under-performing with

	CIFAR 5-Split		CIFAR 10-Split		CIFAR 20-Split		Tiny-ImageNet 40-Split	
Augmentation	BWT	ACC	BWT	ACC	BWT	ACC	BWT	ACC
Without	-2.0 ± 0.5	64.2 ± 1.1	-2.9 ± 0.3	71.2 ± 0.5	-3.7 ± 0.3	79.4 ± 0.6	-4.5 ± 0.3	72.6 ± 0.8
With	-0.2 ± 0.2	80.3 ± 0.6	-0.6 ± 0.2	83.7 ± 0.8	-1.5 ± 0.3	86.6 ± 0.4	-2.1 ± 0.2	78.6 ± 0.6

Table 2. Data augmentation results for LwF with WRN-20-W5 architecture. BWT and ACC in %. All results are averaged over five runs with standard deviations.

			CIFAR 5-Split		CIFAR 10-Split		CIFAR 20-Split		Tiny-ImageNet 40-Split	
Method	Arch.	Aug.	BWT	ACC	BWT	ACC	BWT	ACC	BWT	ACC
EWC	AlexNet-D		$+0.2 \pm 0.1$	58.6 ± 0.9	$+0.7 \pm 0.4$	64.1 ± 0.5	$+0.0 \pm 0.9$	74.0 ± 1.0	-0.8 ± 0.4	63.3 ± 0.9
EWC	AlexNet-D	✓	$+0.0 \pm 0.2$	62.9 ± 1.5	$+0.1 \pm 0.4$	68.4 ± 0.9	-0.5 ± 1.1	75.2 ± 1.3	-1.5 ± 2.0	63.8 ± 2.6
IMM-MEAN	AlexNet-D		-1.2 ± 0.8	58.9 ± 1.1	-0.6 ± 0.7	58.6 ± 1.9	-0.8 ± 0.3	55.9 ± 1.6	-0.6 ± 0.8	43.6 ± 1.3
IMM-MEAN	AlexNet-D	✓	-2.5 ± 1.0	62.5 ± 1.8	-1.3 ± 0.8	61.4 ± 2.0	-1.3 ± 0.5	57.9 ± 2.9	-1.2 ± 0.5	44.7 ± 1.5
IMM-MODE	AlexNet-D		-8.3 ± 1.5	63.7 ± 1.5	-21.7 ± 2.9	58.6 ± 2.9	-30.5 ± 3.2	54.9 ± 3.0	-25.0 ± 1.4	50.6 ± 1.7
IMM-MODE	AlexNet-D	✓	-6.9 ± 0.3	68.9 ± 0.9	-19.8 ± 2.7	64.4 ± 2.9	-31.1 ± 4.2	58.2 ± 4.3	-24.2 ± 2.4	54.6 ± 2.9
HAT	AlexNet-D		$+0.0 \pm 0.0$	67.1 ± 0.6	$+0.0 \pm 0.0$	72.8 ± 0.8	$+0.0 \pm 0.0$	76.6 ± 0.6	$+0.0 \pm 0.0$	65.9 ± 1.1
HAT	AlexNet-D	✓	-0.1 ± 0.0	70.5 ± 0.9	$+0.0 \pm 0.0$	76.2 ± 0.8	$+0.0 \pm 0.0$	78.4 ± 1.0	$+0.0 \pm 0.0$	67.3 ± 0.9
HyperCL	H:Lin,M:RN32		$+0.0 \pm 0.1$	53.0 ± 2.3	$+0.0 \pm 0.0$	62.9 ± 0.4	$+0.0 \pm 0.0$	75.5 ± 1.0	-0.8 ± 0.3	48.9 ± 1.6
HyperCL	H:Lin,M:RN32	✓	$+0.0 \pm 0.0$	69.5 ± 1.1	$+0.0 \pm 0.0$	78.2 ± 0.6	$+0.0 \pm 0.0$	85.3 ± 0.9	-0.9 ± 0.3	60.7 ± 0.3
ACL^o	AlexNet-D**		-	-	-	-	$+0.0 \pm 0.0$	78.0 ± 1.2	-	-
LwF	WRN-20-W5		-2.0 ± 0.5	64.2 ± 1.1	-2.9 ± 0.3	71.2 ± 0.5	-3.7 ± 0.3	79.4 ± 0.6	-4.5 ± 0.3	72.6 ± 0.8
LwF	WRN-20-W5	✓	-0.2 ± 0.2	80.3 ± 0.6	-0.6 ± 0.2	83.7 ± 0.8	-1.5 ± 0.3	86.6 ± 0.4	-2.1 ± 0.2	78.6 ± 0.6
JOINT*	WRN-20-W5		$+4.5 \pm 2.0$	72.3 ± 1.9	$+4.2 \pm 1.9$	80.2 ± 2.0	$+3.0 \pm 1.1$	86.1 ± 0.9	$+3.5 \pm 0.3$	80.3 ± 0.3
JOINT*	WRN-20-W5	✓	$+2.4 \pm 0.8$	85.3 ± 0.5	$+2.3 \pm 0.2$	89.9 ± 0.4	$+1.7 \pm 0.6$	93.2 ± 0.4	$+2.2 \pm 0.5$	86.7 ± 0.4

Table 3. Comparison between multiple methods. BWT and ACC in %. *JOINT does not adhere to the task incremental setup, and is performed in order to serve as the upper bound for LwF. **Slightly different AlexNet-like architecture than used in HAT with a similar capacity. ^{o}results reported in [4]; all other results are reproduced by us and are averaged over five runs with standard deviations. D=Dropout, RN=ResNet, WRN=WideResNet, Lin=a linear layer, H=Hypernetwork, M=Target network.

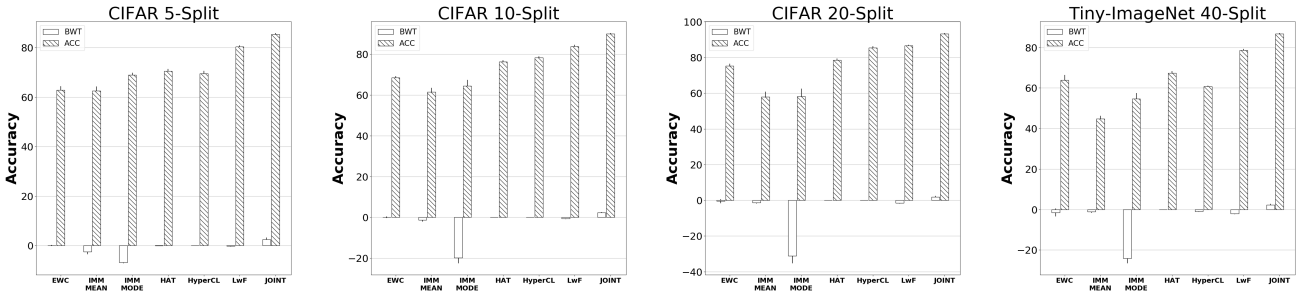
WRN-20-W5, see supplementary. We conjecture that the difference from LwF lies in the type of regularization term used by each method. LwF employs a 'soft' regularization on the network output for previous tasks, which handles statistical shift due to batch normalization better than the weight-based regularization. However, these methods do not suffer from Dropout as they employ hard regularization on the weights which considers their importance. For the comparison table, we use the best evaluated architecture for each method.

All methods, except Hyper-CL and ACL, use separated fully-connected layers with a softmax output for each task as a final layer. Hyper-CL employs a separate generated network for each task, and ACL employs a separate 3-layer MLP with softmax output for each task on top of private and shared concatenation.

Training We made an effort to find the best training protocol for each method, based on the existing literature and initial experiments. For all methods except for Hyper-CL we followed the same training protocol described in Sec. 4.1. For Hyper-CL, we use batch size 32 and with the Adam optimizer [15] with a learning rate of 0.001. As for learning rate scheduling, Hyper-CL uses a validation accuracy to schedule the learning rate by dropping the learning rate with a factor of $(\sqrt{0.1})^{-1}$, if there is no improvement in the validation accuracy for 5 consecutive epochs. The Hyper-CL implementation further employs a custom multi-step scheduler adapted from Keras [2]. However, there is no early stopping in Hyper-CL. Also, no other regularization is used in any of the methods, except to the ones that are inherent to the method itself.

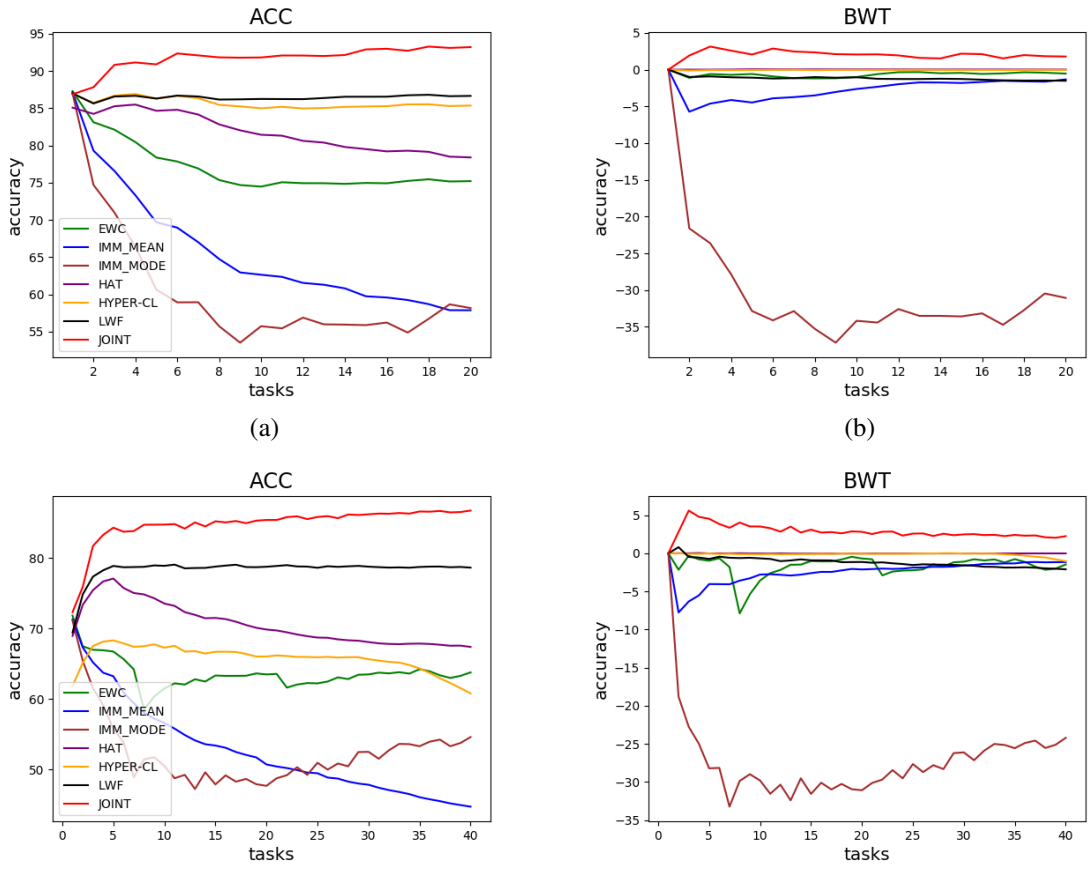
The Hyper-CL official implementation and the author's experiments use the test set for parameter selection in lieu of a proper validation set. We were able to fix and rerun the experiments in time only for the Hyper-CL experiments on CIFAR and not for the Hyper-CL experiments on Tiny-ImageNet. We observed that moving to an independent validation set reduces the performance of Hyper-CL on CIFAR by a significant margin. We, therefore, view the results obtained for this method on Tiny-ImageNet as an upper bound for the method's performance. We note that (i) Hyper-CL is by far the slowest method out of all methods tested, and (ii) On Tiny-ImageNet even though the results of this method are positively biased, the method is not competitive.

The comparison to the literature methods is provided in Tab. 3 and summarized in Fig. 1 for the best configuration



(a) (b) (c) (d)

Figure 1. BWT and ACC of the best performance obtained for each of the evaluated methods average over 5 random seeds. JOINT is an upper-bound training on all past tasks data. (a) CIFAR 5-Split, (b) CIFAR 10-Split, (c) CIFAR 20-Split, (d) Tiny-ImageNet 40-Split.



(a) (b)

(c) (d)

Figure 2. The evolution in time of the accuracy and the forgetting, for the best performing setting of each method average over 5 random seeds. ACC (Eq. 4) after learning task t as a function of t. BWT (Eq. 5) after learning task t as a function of t. (a) & (b) ACC & BWT results over time for CIFAR 20-Split and (c) & (d) similar results over time for Tiny-ImageNet 40-Split.

for each method. Evidently, in contrast to the picture the literature paints, when a proper architecture and added augmentations are used, LwF, which is a simple regularization-base method, outperforms all other methods. The results also show that although IMM has evolved from EWC, both its variants are not competitive with EWC except for the smallest split (CIFAR 5-Split). When considering the augmentation mechanism, we have mixed results. Although augmentations increase ACC, they also increase forgetting for EWC and IMM-MEAN and only slightly reduce forget-

ting for IMM-MODE, which is still quite high. In contrast, for LwF, where we show that augmentations help to both ACC and BWT.

HAT as originally conceived (recall that it is not compatible with ResNets), has a very competitive ACC in CIFAR and even outperforms Hyper-CL for the longer and more challenging sequence of tasks from Tiny-ImageNet. It also further benefits from the augmentation. For Hyper-CL, we can see that although it has a smaller capacity (considering only the hypernetwork learnable parameters for capacity computation) it outperforms all of the baselines for CIFAR when augmentation is used. However, this advantage does not generalize to the Tiny-ImageNet dataset, and it falls behind HAT, and even EWC, for a longer sequence, which further emphasizes the need for comparison over a diverse set of experiments. To check if this shortcoming is a result of the capacity of the model, we experimented with larger models, both for the hypernetwork and target network. We observed that the performance drops significantly in all experiments for the larger network. This result emphasizes the need for careful tuning of the Hyper-CL method, which is challenging since unlike other methods it requires the tuning of two architectures at once, which enlarges the space of possible hyper-parameters dramatically. We note also that [34] reported that out of many architectures tried, the smallest ones showed the best performance-compression ratio.

For ACL, we quote the results for CIFAR 20-Split with no augmentation from the paper itself [4]. The network used in the paper was similar to the one used by HAT. As the results show, ACL outperforms both HAT and Hyper-CL when no augmentation is used. LwF is not considered as a baseline in [4]. However, LwF outperforms ACL with WRN-20-W5 even without augmentation. We emphasize that the difference does not come from capacity, since both networks have a similar capacity as described in Tab. 1.

We further analyze the performance by evaluating ACC and BWT after learning each task. Fig. 2 shows the results for the longer sequences of tasks, 20 for CIFAR and 40 for Tiny-ImageNet (the results for the other experiments can be found in the supplementary). One can observe that the methods differ in substantial ways. First, the non-LwF regularization methods, namely EWC and IMM, are not competitive with LwF since the early stages of the online training. The results also indicate that although more careful tuning between the primary loss and the regularization loss could be made, there is a high degree of trade-off between forgetting and new learning in these methods. Where EWC and IMM-MEAN favor old tasks (low forgetting, low ACC) and IMM-MODE favors new tasks (high forgetting, comparable, or higher, final ACC to IMM-MEAN). Second, the same trade-off exists for HAT: while almost no forgetting exists, the accuracy for new tasks is lower. Since HAT is a parameter isolation method, we conjecture that it struggles to utilize the underlined architecture for learning new tasks. Third, while Hyper-CL and LwF seem close on CIFAR, an important difference is evident in Tiny-ImageNet. Looking at the profile of ACC for Tiny-ImageNet, Fig. 2 (c), shows that Hyper-CL struggles to learn new tasks after task 34 is learned, and the drop of accuracy is not due to forgetting, as is evident by the BWT plot in Fig. 2 (d). Interestingly, this drop also enables EWC to outperform Hyper-CL through more consistent performance after the drop in task 8. Last, for LwF, in both CIFAR and Tiny-ImageNet, it enjoys the capability of learning new tasks and almost does not forget previous tasks. We conclude that, although LwF is a regularization based method, given the right architecture and augmentation, it can maintain both the ability to learn new tasks and to not forget old ones, even at the tails of long tasks sequence.

5. Conclusions

Many of the recent task-incremental publications [21, 29, 1] compare with LwF and found their method to be superior. These conclusions seem to arise from the little incentive authors have to explore the effect of the evaluation settings on prior work, or to invest effort in modernizing the form (*e.g.*, architecture) of baseline methods. However, LwF itself is built on top of solid knowledge-distillation foundations and, as we show, can be upgraded to become extremely competitive.

We demonstrate that the LwF method can benefit from a higher capacity (width-wise) and a network that employs residual connections as well as from augmentations. It is not obvious that the method would benefit from these changes, as many of the other methods cannot benefit from ResNets due to the challenges of applying batch normalization and the need to carefully control the capacity. Moreover, not all methods benefit from augmentations in both ACC and BWT.

Overall, our contributions are two-fold. First, we provide strong baselines for task-incremental methods, that form a solid foundation for comparing future methods. Second, we show the effect of added capacity, residual architectures, and regularization in the form of augmentation on task-incremental methods. Demonstrating sometimes paradoxical behavior, expected to improve performance but deteriorates it. We believe that LwF's ability to benefit from such improvements is a strong indication that this method would stand the test of time.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant ERC CoG 725974).

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2019.

[4] S. Ebrahimi, F. Meier, R. Calandra, Trevor Darrell, and Marcus Rohrbach. Adversarial continual learning. *ArXiv*, abs/2003.09553, 2020.

[5] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[6] Yaroslav Ganin, E. Ustinova, Hana Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17:59:1–59:35, 2016.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[8] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[12] Yen-Chang Hsu, Y. Liu, and Z. Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *ArXiv*, abs/1810.12488, 2018.

[13] Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Dongyan Zhao, Jinwen Ma, and Rui Yan. Overcoming catastrophic forgetting for continual learning via model adaptation. *7th International Conference on Learning Representations, ICLR 2019*, pages 1–13, 2019.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. 2015.

[20] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems*, pages 4652–4662, 2017.

[21] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to Grow: A Continual Structure Learning Framework for Overcoming Catastrophic Forgetting. 2019.

[22] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[23] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[24] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[25] Nick Pawlowski, Andrew Brock, Matthew CH Lee, Martin Rajchl, and Ben Glocker. Implicit weight

uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

[26] Amal Rannen, Rahaf Aljundi, Matthew B Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1320–1328, 2017.

[27] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[28] M. Salzmann, C. Ek, R. Urtasun, and Trevor Darrell. Factorized orthogonal latent spaces. In *AISTATS*, 2010.

[29] Joan Serrà, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[32] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In *Advances in neural information processing systems*, pages 2310–2318, 2013.

[33] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[34] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

[35] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[36] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.