Conditional Channel Gated Networks for Task-Aware Continual Learning

Davide Abati[1]* Jakub Tomczak[2] Tijmen Blankevoort[2] Simone Calderara[1]
Rita Cucchiara[1] Babak Ehteshami Bejnordi[2]

[1]University of Modena and Reggio Emilia
{name.surname}@unimore.it

[2]Qualcomm AI Research†
Qualcomm Technologies Netherlands B.V.
{jtomczak,tijmen,behtesha}@qti.qualcomm.com

Abstract

Convolutional Neural Networks experience catastrophic forgetting when optimized on a sequence of learning problems: as they meet the objective of the current training examples, their performance on previous tasks drops drastically. In this work, we introduce a novel framework to tackle this problem with conditional computation. We equip each convolutional layer with task-specific gating modules, selecting which filters to apply on the given input. This way, we achieve two appealing properties. Firstly, the execution patterns of the gates allow to identify and protect important filters, ensuring no loss in the performance of the model for previously learned tasks. Secondly, by using a sparsity objective, we can promote the selection of a limited set of kernels, allowing to retain sufficient model capacity to digest new tasks. Existing solutions require, at test time, awareness of the task to which each example belongs to. This knowledge, however, may not be available in many practical scenarios. Therefore, we additionally introduce a task classifier that predicts the task label of each example, to deal with settings in which a task oracle is not available. We validate our proposal on four continual learning datasets. Results show that our model consistently outperforms existing methods both in the presence and the absence of a task oracle. Notably, on Split SVHN and Imagenet-50 datasets, our model yields up to 23.98% and 17.42% improvement in accuracy w.r.t. competing methods.

1. Introduction

Machine learning and deep learning models are typically trained offline, by sampling examples independently from the distribution they are expected to deal with at test time. However, when trained online in real-world settings, models may encounter multiple tasks as a sequential stream of activities, without having any knowledge about their relationship or duration in time. Such challenges typically arise in robotics [2], reinforcement learning [29], vision systems [26] and many more (cf. Chapter 4 in [7]). In such scenarios, deep learning models suffer from *catastrophic forgetting* [23, 9], meaning they discard previously acquired knowledge to fit the current observations. The underlying reason is that, while learning the new task, models overwrite the parameters that were critical for previous tasks.

Continual learning research (also called *lifelong* or *incremental* learning) tackles the above mentioned issues [7]. The typical setting considered in the literature is that of a model learning disjoint classification problems one-by-one. Depending on the application requirements, the task for which the current input should be analyzed may or may not be known. The majority of the methods in the literature assume that the label of the task is provided during inference. Such a continual learning setting is generally referred to as task-incremental. In many real-world applications, such as classification and anomaly detection systems, a model can seamlessly instantiate a new task whenever novel classes emerge from the training stream. However, once deployed in the wild, it has to process inputs without knowing in which training task similar observations were encountered. Such a setting, in which task labels are available only during training, is known as class-incremental [35]. Existing methods employ different strategies to mitigate catastrophic forgetting, such as memory buffers [27, 18], knowledge distillation [17], synaptic consolidation [14] and parameters masking [21, 32]. However, recent evidence has shown that existing solutions fail, even for simple datasets, whenever task labels are not available at test time [35].

This paper introduces a solution based on conditional-computing to tackle both task-incremental and class-incremental learning problems. Specifically, our framework relies on separate task-specific classification heads (*multi-head* architecture), and it employs channel-gating [6, 3] in every layer of the (shared) feature extractor. To this aim, we introduce task-dedicated gating modules that dynamically select which filters to apply conditioned on the input feature

*Research conducted during an internship at Qualcomm Technologies Netherlands B.V.

†Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

2575-7075/20/$31.00 ©2020 IEEE
DOI 10.1109/CVPR42600.2020.00399

map. Along with a sparsity objective encouraging the use of fewer units, this strategy enables per-sample model selection and can be easily queried for information about which weights are essential for the current task. Those weights are frozen when learning new tasks, but gating modules can dynamically select to either use or discard them. Contrarily, units that are never used by previous tasks are reinitialized and made available for acquiring novel concepts. This procedure prevents any forgetting of past tasks and allows considerable computational savings in the forward propagation. Moreover, we obviate the need for a task label during inference by introducing a task classifier selecting which classification head should be queried for the class prediction. We train the task classifier alongside the classification heads under the same incremental learning constraints. To mitigate forgetting on the task classification side, we rely on example replay from either episodic or generative memories. In both cases, we show the benefits of performing rehearsal at a task-level, as opposed to previous replay methods that operate at a class-level [27, 5]. To the best of our knowledge, this is the first work that carries out supervised task prediction in a class-incremental learning setting.

We perform extensive experiments on four datasets of increasing difficulty, both in the presence and absence of a task oracle at test time. Our results show that, whenever task labels are available, our model effectively prevents the forgetting problem, and performs similarly to or better than state-of-the-art solutions. In the task agnostic setting, we consistently outperform competing methods.

2. Related work

Continual learning. Catastrophic forgetting has been a well-known problem of neural networks [23]. Early approaches to alleviate the issue involved orthogonal representation learning and replay of prior samples [9]. The recent advent in deep learning has led to the widespread use of deep neural networks in the continual learning field. First attempts, such as Progressive Neural Networks [30] tackle the forgetting problem by introducing a new set of parameters for each new task at the expense of limited scalability. Another popular solution is to apply knowledge distillation by using the past parametrizations of the model as a reference when learning new tasks [17].

Consolidation approaches emerged recently with the focus of identifying the weights that are critically important for prior tasks and preventing significant updates to them during the learning of new tasks. The relevance/importance estimation for each parameter can be carried out through the Fisher Information Matrix [14], the path integral of loss gradients [39], gradient magnitude [1] and a posteriori uncertainty estimation in a Bayesian Neural Network [25].

Other popular consolidation strategies rely on the estimation of binary masks that directly map each task to the set of parameters responsible for it. Such masks can be estimated either by random assignment [22], pruning [21] or gradient descent [20, 32]. However, existing mask-based approaches can only operate in the presence of an oracle providing the task label. Our work is akin to the above-mentioned models, with two fundamental differences: i) our binary masks (gates) are dynamically generated and depend on the network input, and ii) we promote mask-based approaches to class-incremental learning settings, by relying on a novel architecture comprising a task classifier.

Several models allow access to a finite-capacity memory buffer (*episodic* memory), holding examples from prior tasks. A popular approach is iCaRL [27], which computes class prototypes as the mean feature representation of stored memories, and classifies test examples in a nearest-neighbor fashion. Alternatively, other approaches intervene in the training algorithm, proposing to adjust the gradient computed on the current batch towards an update direction that guarantees non-destructive effects on the stored examples [18, 5, 28]. Such an objective can imply the formalization of constrained optimization problems [18, 5] or the employment of meta-learning algorithms [28]. Differently, *generative* memories do not rely on the replay of any real example whatsoever, in favor of generative models from which fake examples of past tasks can be efficiently sampled [34, 38, 26]. In this work, we also rely on either episodic or generative memories to deal with the class-incremental learning setting. However, we carry out replay only to prevent forgetting of the task predictor, thus avoiding to update task-specific classification heads.

Conditional computation. Conditional computation research focuses on deep neural networks that adapt their architecture to the given input. Although the first work has been applied to language modeling [33], several works applied such concept to computer vision problems. In this respect, prior works employ binary gates deciding whether a computational block has to be executed or skipped. Such gates may either drop entire residual blocks [36, 37] or specific units within a layer [6, 3]. In our work, we rely on the latter strategy, learning a set of task-specific gating modules selecting which kernels to apply on the given input. To our knowledge, this is the first application of data-dependent channel-gating in continual learning.

3. Model

3.1. Problem setting and objective

We are given a parametric model, i.e., a neural network, called a *backbone* or *learner* network, which is exposed to a sequence of N tasks to be learned, $\mathcal{T} = \{T_1, \ldots, T_N\}$. Each task T_i takes the form of a classification problem, $T_i = \{\mathbf{x}_j, y_j\}_{j=1}^{n_i}$, where $\mathbf{x}_j \in \mathbb{R}^m$ and $y_j \in \{1, \ldots, C_i\}$.

A task-incremental setting requires to optimize:

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{t}\sim\mathcal{T}}\left[\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim T_{\mathbf{t}}}\left[\log p_{\theta}(\mathbf{y}|\mathbf{x},\mathbf{t})\right]\right], \tag{1}$$

where θ identifies the parametrization of the learner network, and $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{t}$ are random variables associated with the observation, the label and the task of each example, respectively. Such a maximization problem is subject to the continual learning constraints: as the model observes tasks sequentially, the outer expectation in Eq. 1 is troublesome to compute or approximate. Notably, this setting requires the assumption that the identity of the task each example belongs to is known at both training and test stages. Such information can be exploited in practice to isolate relevant output units of the classifier, preventing the competition between classes belonging to different tasks through the same softmax layer (*multi-head*).

Class-incremental models solve the following optimization:

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{t}\sim\mathcal{T}}\left[\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim T_{\mathbf{t}}}\left[\log p_{\theta}(\mathbf{y}|\mathbf{x})\right]\right]. \tag{2}$$

Here, the absence of task conditioning prevents any form of task-aware reasoning in the model. This setting requires to merge the output units into a single classifier (*single-head*) in which classes from different tasks compete with each other, often resulting in more severe forgetting [35]. Although the model could learn based on task information, this information is not available during inference.

To deal with observations from unknown tasks, while retaining advantages of multi-head settings, we will jointly optimize for class as well as task prediction, as follows:

$$\begin{aligned}\max_{\theta} \quad & \mathbb{E}_{\mathbf{t}\sim\mathcal{T}}\left[\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim T_{\mathbf{t}}}\left[\log p_{\theta}(\mathbf{y},\mathbf{t}|\mathbf{x})\right]\right] = \\ & \mathbb{E}_{\mathbf{t}\sim\mathcal{T}}\left[\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim T_{\mathbf{t}}}\left[\log p_{\theta}(\mathbf{y}|\mathbf{x},\mathbf{t}) + \log p_{\theta}(\mathbf{t}|\mathbf{x})\right]\right].\end{aligned} \tag{3}$$

Eq. 3 describes a twofold objective. On the one hand, the term $\log p(\mathbf{y}|\mathbf{x},\mathbf{t})$ is responsible for the *class classification given the task*, and resembles the multi-head objective in Eq. 1. On the other hand, the term $\log p(\mathbf{t}|\mathbf{x})$ aims at *predicting the task* from the observation. This prediction relies on a task classifier, which is trained incrementally in a single-head fashion. Notably, the objective in Eq. 3 shifts the single-head complexities from a class prediction to a task prediction level, with the following benefits:

- given the task label, there is no drop in class prediction accuracy;
- classes from different tasks never compete with each other, neither during training nor during test;
- the challenging single-head prediction step is shifted from class to task level; as tasks and classes form a two-level hierarchy, the prediction of the former is arguably easier (as it acts at a coarser semantic level).

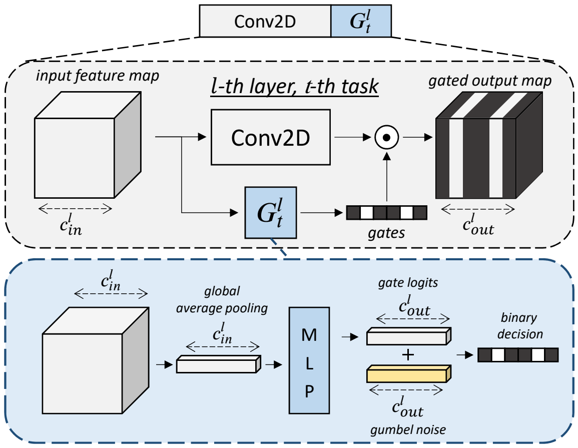

Figure 1: The proposed gating scheme for a convolution layer. Depending on the input feature map, the gating module G_t^l decides which kernels should be used.

3.2. Multi-head learning of class labels

In this section, we introduce the conditional computation model we used in our work. Fig. 1 illustrates the gating mechanism used in our framework. We limit the discussion of the gating mechanism to the case of convolutional layers, as it also applies to other parametrized mappings such as fully connected layers or residual blocks. Consider $\mathbf{h}^l \in \mathbb{R}^{c_{in}^l,h,w}$ and $\mathbf{h}^{l+1} \in \mathbb{R}^{c_{out}^l,h',w'}$ to be the input and output feature maps of the l-th convolutional layer respectively. Instead of $\mathbf{h}^{l+1}$, we will forward to the following layer a sparse feature map $\hat{\mathbf{h}}^{l+1}$, obtained by pruning uninformative channels. During the training of task t, the decision regarding which channels have to be activated is delegated to a gating module G_t^l, that is conditioned on the input feature map $\mathbf{h}^l$:

$$\hat{\mathbf{h}}^{l+1} = G_t^l(\mathbf{h}^l) \odot \mathbf{h}^{l+1}, \tag{4}$$

where $G_t^l(\mathbf{h}^l) = [g_1^l, \ldots, g_{c_{out}^l}^l]$, $g_i^l \in \{0,1\}$, and $\odot$ refers to channel-wise multiplication. To be compliant with the incremental setting, we instantiate a new gating module each time the model observes examples from a new task. However, each module is designed as a light-weight network with negligible computation costs and number of parameters. Specifically, each gating module comprises a Multi-Layer Perceptron (MLP) with a single hidden layer featuring 16 units, followed by a batch normalization layer [12] and a ReLU activation. A final linear map provides log-probabilities for each output channel of the convolution.

Back-propagating gradients through the gates is challenging, as non-differentiable thresholds are employed to take binary on/off decisions. Therefore, we rely on the Gumbel-Softmax sampling [13, 19], and get a biased estimate of the gradient utilizing the straight-through estimator [4]. Specif-

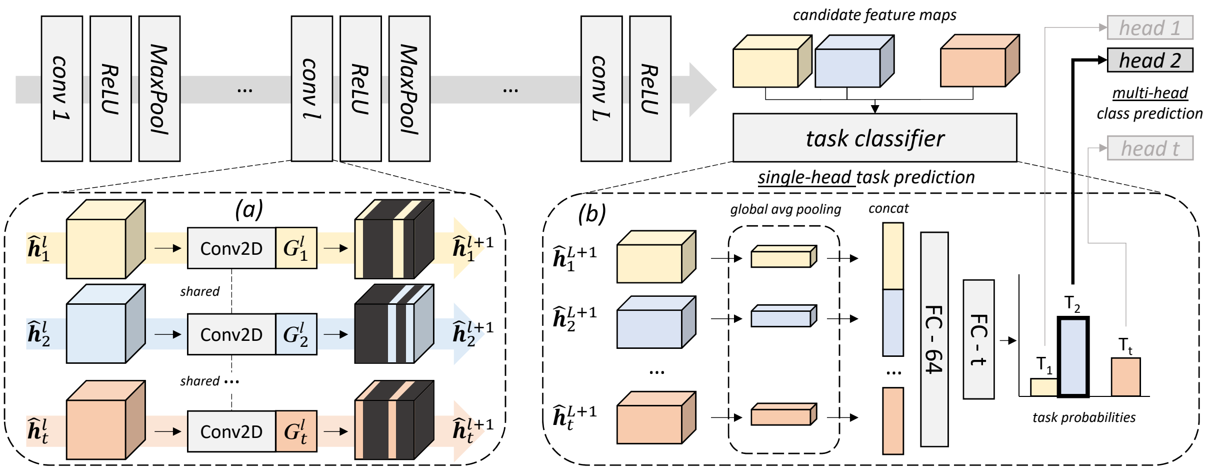

Figure 2: Illustration of the task prediction mechanism for a generic backbone architecture. First (block 'a'), the l-th convolutional layer is fed with multiple gated feature maps, each of which is relevant for a specific task. Every feature map is then convolved with kernels selected by the corresponding gating module G^l_x, and forwarded to the next module. At the end of the network the task classifier (block 'b') takes as input candidate feature maps and decides which task to solve.

ically, we employ the hard threshold in the forward pass (zero-centered) and the sigmoid function in the backward pass (with temperature $\tau = 2/3$).

Moreover, we penalize the number of active convolutional kernels with the sparsity objective:

$$\mathcal{L}_{sparse} = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim T_\mathbf{t}} \left[\frac{\lambda_s}{L} \sum_{l=1}^{L} \frac{\|G^l_t(\mathbf{h}^l)\|_1}{c^l_{out}} \right], \tag{5}$$

where L is the total number of gated layers, and λ_s is a coefficient controlling the level of sparsity. The sparsity objective instructs each gating module to select a minimal set of kernels, allowing us to conserve filters for the optimization of future tasks. Moreover, it allows us to effectively adapt the capacity of the allocated network depending on the difficulty of the task and the observation at hand. Such a data-driven model selection contrasts with other continual learning strategies that employ fixed ratios for model growing [30] or weight pruning [21].

At the end of the optimization for task t, we compute a relevance score r^l_k for each unit in the l-th layer by estimating the firing probability of their gates on a validation set $T^{val}_\mathbf{t}$:

$$r^{l,t}_k = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim T^{val}_\mathbf{t}}[p(\mathbb{I}[g^l_k = 1])], \tag{6}$$

where $\mathbb{I}[\cdot]$ is an indicator function, and $p(\cdot)$ denotes a probability distribution. By thresholding such scores, we obtain two sets of kernels. On the one hand, we *freeze relevant kernels* for the task t, so that they will be available but not updatable during future tasks. On the other hand, we *re-initialize non-relevant kernels*, and leave them learnable by subsequent tasks. In all our experiments, we use a threshold equal to 0, which prevents any forgetting at the expense of a reduced model capacity left for future tasks.

Note that within this framework, it is trivial to monitor the number of learnable units left in each layer. As such, if the capacity of the backbone model saturates, we can quickly grow the network to digest new tasks. However, because the gating modules of new tasks can dynamically choose to use previously learned filters (if relevant for their input), learning of new tasks generally requires less learnable units. In practice, we never experienced the saturation of the backbone model for learning new tasks. Apart from that, because of our conditional channel-gated network design, increasing the model capacity for future tasks will have minimal effects on the computation cost at inference, as reported by the analysis in Sec. 4.5.

3.3. Single-head learning of task labels

The gating scheme presented in Sec. 3.2 allows the immediate identification of important kernels for each past task. However, it cannot be applied in the task-agnostic setting as is, since it requires the knowledge about which gating module G^l_x has to be applied for layer l, where $x \in \{1, \dots, t\}$ represents the unknown task. Our solution is to employ all gating modules $[G^l_1, \dots, G^l_t]$, and to propagate all gated layer outputs $[\hat{\mathbf{h}}^{l+1}_1, \dots, \hat{\mathbf{h}}^{l+1}_t]$ forward. In turn, the following layer $l+1$ receives the list of gated outputs from layer l, applies its gating modules $[G^{l+1}_1, \dots, G^{l+1}_t]$ and yields the list of outputs $[\hat{\mathbf{h}}^{l+2}_1, \dots, \hat{\mathbf{h}}^{l+2}_t]$. This mechanism generates parallel streams of computation in the network, sharing the same layers but selecting different sets of units to activate for each of them (Fig. 2). Despite the fact that the number of parallel streams grows with the number of tasks, we found our solution to be computationally cheaper than the backbone network (see Sec. 4.5). This is because of the gating modules which select a limited number of convolutional filters in each stream.

After the last convolutional layer, indexed by L, we are given a list of t candidate feature maps $[\hat{\mathbf{h}}_1^{L+1}, \dots, \hat{\mathbf{h}}_t^{L+1}]$ and as many classification heads. The task classifier is fed with a concatenation of all feature maps:

$$h = \bigoplus_{i=1}^{t} [\mu(\hat{\mathbf{h}}_i^{L+1})], \qquad (7)$$

where μ denotes the global average pooling operator over the spatial dimensions and $\bigoplus$ describes the concatenation along the feature axis. The architecture of the task classifier is based on a shallow MLP with one hidden layer featuring 64 ReLU units, followed by a softmax layer predicting the task label. We use the standard cross-entropy objective to train the task classifier. Optimization is carried out jointly with the learning of class labels at task t. Thus, the network not only learns features to discriminate the classes inside task t, but also to allow easier discrimination of input data from task t against all prior tasks.

The single-head task classifier is exposed to catastrophic forgetting. Recent papers have shown that replay-based strategies represent the most effective continual learning strategy in single-head settings [35]. Therefore, we choose to ameliorate the problem by rehearsal. In particular, we consider the following approaches.

Episodic memory. A small subset of examples from prior tasks is used to rehearse the task classifier. During the training of task t, the buffer holds C random examples from past tasks $1, \dots, t-1$ (where C denotes a fixed capacity). Examples from the buffer and the current batch (from task t) are re-sampled so that the distribution of task labels in the rehearsal batch is uniform. At the end of task t, the data in the buffer is subsampled so that each past task holds $m = C/t$ examples. Finally, m random examples from task t are selected for storage.

Generative memory. A generative model is employed for sampling fake data from prior tasks. Specifically, we utilize Wasserstein GANs with Gradient Penalty (WGAN-GP [10]). To overcome forgetting in the sampling procedure, we use multiple generators, each of which models the distribution of examples of a specific task.

In both cases, replay is only employed for rehearsing the task classifier and not the classification heads. To summarize, the complete objective of our model includes: the cross-entropy at a class level ($p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{t})$ in Eq. 3), the cross-entropy at a task level ($p_\theta(\mathbf{t}|\mathbf{x})$ in Eq. 3) and the sparsity term ($\mathcal{L}_{sparse}$ in Eq. 5).

4. Experiments

4.1. Datasets and backbone architectures

We experiment with the following datasets:

- Split MNIST: the MNIST handwritten classification benchmark [16] is split into 5 subsets of consecutive classes. This results into 5 binary classification tasks that are observed sequentially.
- Split SVHN: the same protocol applied as in Split MNIST, but employing the SVHN dataset [24].
- Split CIFAR-10: the same protocol applied as in Split MNIST, but employing the CIFAR-10 dataset [15].
- Imagenet-50 [26]: a subset of the iILSVRC-2012 dataset [8] containing 50 randomly sampled classes and 1300 images per category, split into 5 consecutive 10-way classification problems. Images are resized to a resolution of 32x32 pixels.

As for the backbone models, for the MNIST and SVHN benchmarks, we employ a three-layer CNN with 100 filters per layer and ReLU activations (*SimpleCNN* in what follows). All convolutions except for the last one are followed by a 2x2 max-pooling layer. Gating is applied after the pooling layer. A final global average pooling followed by a linear classifier yields class predictions. For the CIFAR-10 and Imagenet-50 benchmarks we employed a ResNet-18 [11] model as backbone. The gated version of a ResNet basic block is represented in Fig. 3. As illustrated, two independent sets of gates are applied after the first convolution and after the residual connection, respectively.

All models were trained with SGD with momentum until convergence. After each task, model selection is performed for all models by monitoring the corresponding objective on a held-out set of examples from the current task (i.e., we don't rely on examples of past tasks for validation purposes). We apply the sparsity objective introduced in Sec. 3.2 only after a predetermined number of epochs, to provide the model the possibility to learn meaningful kernels before starting pruning the uninformative ones. We refer to the supplementary material for further implementation details.

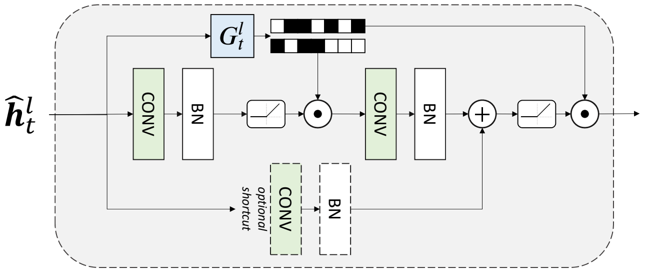

Figure 3: The gating scheme applied to ResNet-18 blocks. Gating on the *shortcut* is only applied when downsampling.

	Split MNIST						Split SVHN						Split CIFAR-10					
	T_1	T_2	T_3	T_4	T_5	avg	T_1	T_2	T_3	T_4	T_5	avg	T_1	T_2	T_3	T_4	T_5	avg
Joint (UB)	0.999	0.999	0.999	1.000	0.995	0.999	0.983	0.972	0.982	0.983	0.941	0.972	0.996	0.964	0.979	0.995	0.983	0.983
EWC-On	0.971	0.994	0.934	0.982	0.932	0.963	0.906	0.966	0.967	0.965	0.889	0.938	0.758	0.804	0.803	0.952	0.960	0.855
LwF	0.998	0.979	0.997	**0.999**	0.985	0.992	0.974	0.928	0.863	0.832	0.513	0.822	0.948	0.873	0.671	0.505	0.514	0.702
HAT	0.999	**0.996**	0.999	0.998	0.990	**0.997**	0.971	0.967	0.970	0.976	0.924	0.962	0.988	0.911	**0.953**	**0.985**	0.977	0.963
ours	**1.00**	0.994	**1.00**	**0.999**	**0.993**	**0.997**	**0.978**	**0.972**	**0.983**	**0.988**	**0.946**	**0.974**	**0.994**	**0.917**	0.950	0.983	**0.978**	**0.964**

Table 1: Task-incremental results. For each method, we report the final accuracy on all task after incremental training.

4.2. Task-incremental setting

In the task-incremental setting, an oracle can be queried for task labels during test time. Therefore, we don't rely on the task classifier, exploiting ground-truth task labels to select which gating modules and classification head should be active. This section validates the suitability of the proposed data-dependent gating scheme for continual learning. We compare our model against several competing methods:

- *Joint*: the backbone model trained jointly on all tasks while having access to the entire dataset. We considered its performance as the upper bound.
- *Ewc-On* [31]: the online version of Elastic Weight Consolidation, relying on the latest MAP estimate of the parameters and a running sum of Fisher matrices.
- *LwF* [17]: an approach in which the task loss is regularized by a distillation objective, employing the initial state of the model on the current task as a teacher.
- *HAT* [32]: a mask-based model conditioning the active units in the network on the task label. Despite being the most similar approach to our method, it can only be applied in task-incremental settings.

Tab. 1 reports the comparison between methods, in terms of accuracy on all tasks after the whole training procedure. Despite performing very similarly for MNIST, the gap in the consolidation capability of different models emerges as the dataset grows more and more challenging. It is worth mentioning several recurring patterns. First, LwF struggles when the number of tasks grows larger than two. Although its distillation objective is an excellent regularizer against forgetting, it does not allow enough flexibility to the model to acquire new knowledge. Consequently, its accuracy on the most recent task gradually decreases during sequential learning, whereas the performance on the first task is kept very high. Moreover, results highlight the suitability of gating-based schemes (HAT and ours) with respect to other consolidation strategies such as EWC Online. Whereas the former ones prevent any update of relevant parameters, the latter approach only penalizes updating them, eventually incurring a significant degree of forgetting. Finally, the table shows that our model either performs on-par or outperforms HAT on all datasets, suggesting the beneficial effect of our data-dependent gating scheme and sparsity objective.

4.3. Class-incremental with episodic memory

Next, we move to a class-incremental setting in which no awareness of task labels is available at test time, significantly increasing the difficulty of the continual learning problem. In this section, we set up an experiment for which the storage of a limited amount of examples (buffer) is allowed. We compare against:

- Full replay: upper bound performance given by replay to the network of an unlimited number of examples.
- iCaRL [27] an approach based on a nearest-neighbor classifier exploiting examples in the buffer. We report the performances both with the original buffer-filling strategy (*iCaRL-mean*) and with the randomized algorithm used for our model (*iCaRL-rand*);
- A-GEM [5]: a buffer-based method correcting parameter updates on the current task so that they don't contradict the gradient computed on the stored examples.

Results are summarized in Fig. 4, illustrating the final average accuracy on all tasks at different buffer sizes for the class-incremental Split-MNIST and Split-SVHN benchmarks. The figure highlights several findings. Surprisingly, A-GEM yields a very low performance on MNIST, while providing higher results on SVHN. Further examination on the former dataset revealed that it consistently reaches competitive accuracy on the most recent task, while mostly forgetting the prior ones. The performance of iCaRL, on the other hand, does not seem to be significantly affected by changing its buffer filling strategy. Moreover, its accuracy seems not to scale with the number of stored examples. In contrast to these methods, our model primarily utilizes the few stored examples for the rehearsal of coarse-grained task prediction, while retaining the accuracy of fine-grained class prediction. As shown in Fig. 4, our approach consistently outperforms competing approaches in the class-incremental setting with episodic memory.

4.4. Class-incremental with generative memory

Next, we experiment with a class-incremental setting in which no examples are allowed to be stored whatsoever. A popular strategy in this framework is to employ generative models to approximate the distribution of prior tasks and

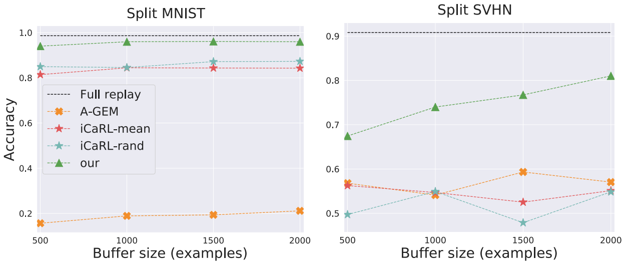

Figure 4: Final mean accuracy on all tasks when an episodic memory is employed, as a function of the buffer capacity.

rehearse the backbone network by sampling fake observations from them. Among these, DGM [26] is the state-of-the-art approach, which proposes a class-conditional GAN architecture paired with a hard attention mechanism similar to the one of HAT [32]. Fake examples from the GAN generator are replayed to the discriminator, which includes an auxiliary classifier providing a class prediction. As for our model, as mentioned in Sec. 3.3, we rely on multiple task-specific generators. For a detailed discussion of the architecture of the employed WGANs, we refer the reader to the supplementary material. Tab. 2 compares the results of DGM and our model for the class-incremental setting with generative memory. Once again, our method of exploiting rehearsal for only the task classifier proves beneficial. DGM performs particularly well on Split MNIST, where hallucinated examples are almost indistinguishable from real examples. On the contrary, results suggest that class-conditional rehearsal becomes potentially unrewarding as the complexity of the modeled distribution increases, and the visual quality of generated samples degrades.

	MNIST	SVHN	CIFAR-10	Imagenet-50
DGMw [26]	0.9646	0.7438	0.5621	0.1782
DGMa [26]	**0.9792**	0.6689	0.5175	0.1516
ours	0.9727	**0.8341**	**0.7006**	**0.3524**

Table 2: Class-incremental continual learning results, when replayed examples are provided by a generative model.

4.5. Model analysis

Episodic vs. generative memory. To understand which rehearsal strategy has to be preferred when dealing with class-incremental learning problems, we raise the following question: What is more beneficial between a limited amount of real examples and a (potentially) unlimited amount of generated examples? To shed light on this matter, we report our models' performances on Split SVHN and Split CIFAR-10 as a function of memory budget. Specifically, we compute the memory consumption of episodic memories as the cumulative size of the stored examples. As for generative memories, we consider the number of bytes needed to store their parameters (in single-precision floating-point format), discarding the corresponding discriminators as well as inner activations generated in the sampling process. Fig. 5 presents the result of the analysis. As can be seen, the variant of our model relying on memory buffers consistently outperforms its counterpart relying on generative modeling. In the case of CIFAR-10, the generative replay yields an accuracy comparable with an episodic memory of ≈ 1.5 MBs, which is more than 20 times smaller than its generators. The gap between the two strategies shrinks on SVHN, due to the simpler image content resulting in better samples from the generators. Finally, our method, when based on memory buffers, outperforms the DGMw model [26] on Split-SVHN, albeit requiring 3.6 times less memory.

Gate analysis. We provide a qualitative analysis of the activation of gates across different tasks in Fig. 6. Specifically, we use the validation sets of Split MNIST and Imagenet-50 to compute the probability of each gate to be triggered by images from different tasks[1]. The analysis of the figure suggests two pieces of evidence: First, as more tasks are observed, previously learned features are re-used. This pattern shows that the model does not fall into degenerate solutions, e.g., by completely isolating tasks into different sub-networks. On the contrary, our model profitably exploits pieces of knowledge acquired from previous tasks for the optimization of the future ones. Moreover, a significant number of gates never fire, suggesting that a considerable portion of the backbone capacity is available for learning even more tasks. Additionally, we showcase how images from different tasks activating the same filters show some resemblance in low-level or semantic features (see the caption for details).

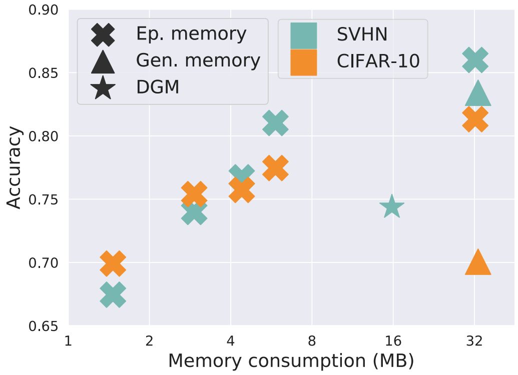

Figure 5: Accuracy as a function of replay memory budget.

[1] we report such probabilities for specific layers: layer 1 for Split MNIST (Simple CNN), block 5 for Imagenet-50 (ResNet-18).

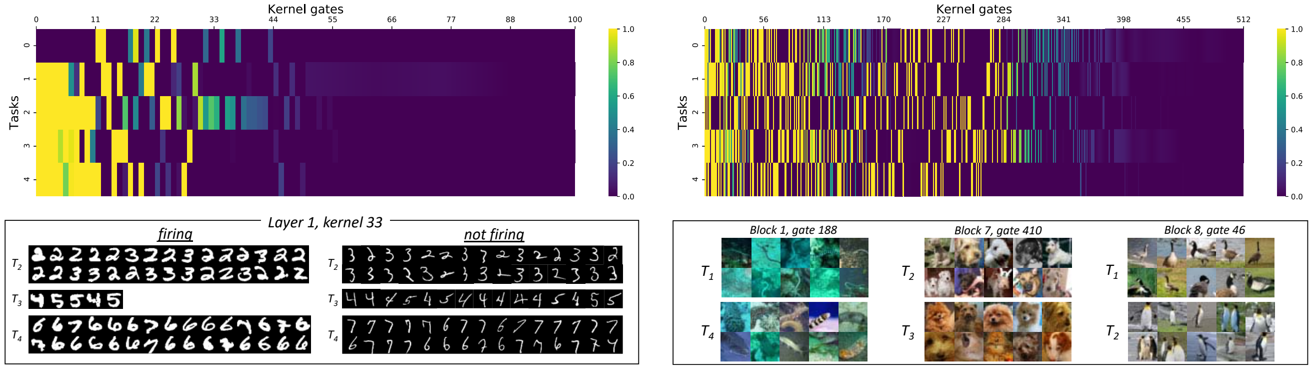

Figure 6: Illustration of the gate execution patterns for continually trained models on MNIST (left) and Imagenet-50 (right) datasets. The histograms in the top left and top right show the firing probability of gates in the 1st layer and the 5th residual block respectively. For better illustration, gates are sorted by overall execution rate over all tasks. The bottom-left box shows images from different tasks either triggering or not triggering a specific gate on Split MNIST. The bottom-right box illustrates how - on Imagenet-50 - correlated classes from different tasks fire the same gates (e.g., fishes, different breeds of dogs, birds).

On the cost of inference. We next measure the inference cost of our model as the number of tasks increases. Tab. 3 reports the average number of multiply-add operations (MAC count) of our model on the test set of Split MNIST and Split CIFAR-10 after learning each task. Moreover, we report the MACs of HAT [32] as well as the cost of forward propagation in the backbone network (i.e. the cost of any other competing method mentioned it this section). In the task-incremental setting, our model obtains a meaningful saving in the number of operations, thanks to the data-dependent gating modules selecting only a small subset of filters to apply. In contrast, forward propagation in a class-incremental setting requires as many computational streams as the number of tasks observed so far. However, each of them is extremely cheap as few convolutional units are active. As presented in the table, also in the class-incremental setting, the number of operations never exceeds the cost of forward propagation in the backbone model. The reduction in inference cost is particularly significant for Split CIFAR-10, which is based on a ResNet-18 backbone.

	Split MNIST			Split CIFAR-10		
	(Simple CNN)			*(ResNet-18)*		
	HAT TI	our TI	our CI	HAT TI	our TI	our CI
Up to T_1	0.151	0.064	0.064	31.937	2.650	2.650
Up to T_2	0.168	0.101	0.209	32.234	4.628	9.199
Up to T_3	0.194	0.137	0.428	36.328	5.028	15.024
Up to T_4	0.221	0.136	0.559	38.040	5.181	20.680
Up to T_5	0.240	0.142	0.725	39.835	5.005	24.927
backbone		0.926			479.920	

Table 3: Average MAC counts ($\times 10^6$) of inference in Split MNIST and Split CIFAR-10. We compute MACs on the test sets, at different stages of the optimization (up to T_t), both in task-incremental (TI) and class-incremental (CI) setups.

Limitations and future works. Training our model can require a lot of GPU memory for bigger backbones. However, by exploiting the inherent sparsity of activation maps, several optimizations are possible. Secondly, we expect the task classifier to be susceptible to the degree of semantic separation among tasks. For instance, a setting where tasks are semantically well-defined, like T_1 = {cat,dog}, T_2 = {car,truck} (*animals / vehicles*), should favor the task classifier with respect to its transpose T_1 = {cat,car}, T_2 = {dog,truck}. However, we remark that in our experiments the assigment of classes to tasks is always random. Therefore, our model could perform even better in the presence of coherent tasks.

5. Conclusions

We presented a novel framework based on conditional computation to tackle catastrophic forgetting in convolutional neural networks. Having task-specific light-weight gating modules allows us to prevent catastrophic forgetting of previously learned knowledge. Besides learning new features for new tasks, the gates allow for dynamic usage of previously learned knowledge to improve performance. Our method can be employed both in the presence and in the absence of task labels during test. In the latter case, a task classifier is trained to take the place of a task oracle. Through extensive experiments, we validated the performance of our model against existing methods both in task-incremental and class-incremental settings and demonstrated state-of-the-art results in four continual learning datasets.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *European Conference on Computer Vision*, 2018. 2

[2] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2019. 1

[3] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaped channel gated networks. *International Conference on Learning Representations*, 2020. 1, 2

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[5] Arslan Chaudhry, MarcAurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2019. 2, 6

[6] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gaternet for dynamic filter selection in cnns. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2019. 1, 2

[7] Zhiyuan Chen and Bing Liu. *Lifelong machine learning*. Morgan & Claypool Publishers, 2018. 1

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2009. 5

[9] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 1999. 1, 2

[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Neural Information Processing Systems*, 2017. 5

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2016. 5

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 2015. 3

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations*, 2017. 3

[14] 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*, 2017. 1, 2

[15] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5

[16] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits, 1998. 5

[17] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*. Springer, 2016. 1, 2, 6

[18] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Neural Information Processing Systems*, 2017. 1, 2

[19] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations*, 2017. 3

[20] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *European Conference on Computer Vision*, 2018. 2

[21] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2018. 1, 2, 4

[22] Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 2018. 2

[23] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Elsevier, 1989. 1, 2

[24] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *Neural Information Processing Systems Workshops*, 2011. 5

[25] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *International Conference on Learning Representations*, 2018. 2

[26] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2019. 1, 2, 5, 7

[27] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2017. 1, 2, 6

[28] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *International Conference on Learning Representations*, 2019. 2

[29] Mark B Ring. CHILD: A first step towards continual learning. *Machine Learning*, 1997. 1

[30] 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. 2, 4

[31] Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable

framework for continual learning. *International Conference on Machine Learning*, 2018. 6

[32] Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *International Conference on Machine Learning*, 2018. 1, 2, 6, 7, 8

[33] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *International Conference on Learning Representations*, 2017. 2

[34] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Neural Information Processing Systems*, 2017. 2

[35] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *Neural Information Processing Systems Workshops*, 2018. 1, 3, 5

[36] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *European Conference on Computer Vision*, 2018. 2

[37] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision*, 2018. 2

[38] Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. In *Neural Information Processing Systems*, 2018. 2

[39] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2017. 2