Continual Learning via Local Module Composition

Anonymous Author(s)
Affiliation
Address
email

Abstract

Modularity is a compelling solution to continual learning (CL), the problem of modelling sequences of related tasks. Learning and then composing modules to solve different tasks provides an abstraction to address the principal challenges of CL including catastrophic forgetting, backward and forward transfer across tasks, and sub-linear model growth. We introduce local module composition (LMC), an approach to modular CL where each module is endowed with a local structural component that estimates modules' relevance to the input. Dynamic module composition is performed layer-wise based on local relevance scores and does not require task identities. This makes LMC more generally applicable than current modular CL methods, which rely on task identities. In addition to module selection and learning rules, LMC also tracks statistics about the input distribution to add new modules when outlier samples are detected. In a first set of experiments, LMC performs favorably compared to existing methods on the recent Continual Transfer-learning Benchmark without requiring task identities. In another study, we show the ability of LMC to generalize to related but unseen tasks (OOD) as well as to compose independently trained models without fine-tuning. In a final continual meta-learning study, we show that LMC performs as well as the strong replay baseline on the Omniglot-MNIST-fMNIST (OMF) and mini-Imagenet environment sequences. LMC outperforms the baseline methods when evaluated on mixtures of tasks containing samples from all datasets seen during continual meta-training on OMF sequences.

1 Introduction

The goal of continual learning (CL) is to learn efficiently from a non-stationary stream of tasks without (catastrophically) forgetting previous tasks [48]. CL is often modelled as a trade-off between knowledge retention (stability) and knowledge expansion (plasticity) [22, 50]. Parameter sharing can provide control over this trade-off. For example, learning a single model shared across tasks results in better knowledge transfer and faster learning at the expense of forgetting [36, 45]. Conversely, learning a separate model per task eliminates forgetting, but minimizes transfer and data efficiency [2, 33].

Modular learning aims at balancing transfer and forgetting by learning a set of specialized modules that can be recomposed to solve (new) tasks while only updating a subset of relevant modules or adding new modules [6, 37, 23]. In principle, a modular learner capable of composing modules in meaningful structures can provide additional benefits including **(i)** compute gains due to only executing modules that are relevant to a task [37, 4]; **(ii)** memory gains due to instantiating a number of modules sub-linearly w.r.t. the number of tasks; **(iii)** systematic [8] and out-of-distribution (OOD) generalization [17] through knowledge recombination; and **(iv)** biological plausibility [72, 71, 76].

Designing modular methods for CL comes with two main challenges. The first is *how and when to add new modules* to ensure sufficient plasticity learn new tasks. Existing modular methods use greedy search variants, expanding the model when it improves validation performance [73, 49]. The

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

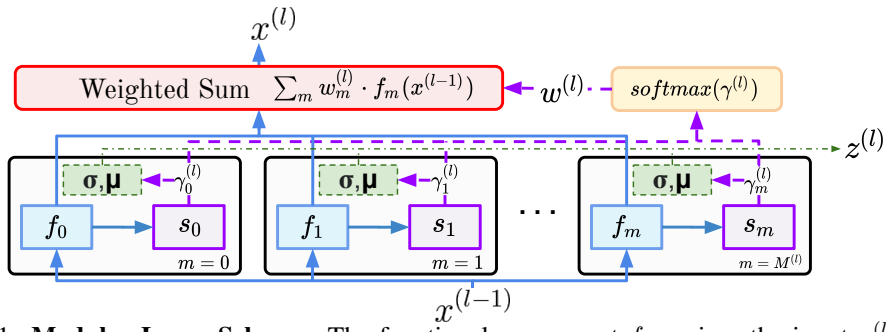


Figure 1: **Modular Layer Scheme**. The functional component f receives the input $x^{(l-1)}$ and feeds it to structural component s. The output of the structural component is used to calculate the importance score $\gamma_m^{(l)}$ using Eq. 5, which are normalized to obtain the attention vector $w^{(l)}$. The layer output is the weighted sum of the functional outputs of each module. μ and σ are the running mean and variance of the scores $\gamma_m^{(l)}$ used to detect outlier inputs and to trigger module addition.

second challenge is *how to compose*, that is retrieve task-specific structural knowledge given a new task (previously seen or not). Existing methods rely on a task's identifier (ID) to retrieve task-specific structural knowledge, which comes either in form of an optimal module layout [73] or as a model- and task-specific controller network that generates modular layouts [49]. Unfortunately, in many realistic CL scenarios task identities are unavailable at test time [28, 12]. Lifting this limitation is challenging since standard mechanisms for task inference, for example leveraging a task-inference model, could be subject to forgetting themselves.

To address both challenges we equip each module with a *local structural component* that predicts a score indicating how relevant the module is for a given input. In-distribution inputs result in high scores while out-of-distribution inputs result in low scores. In other words, modules self-determine their relevance given an input. This local component is used for composing modules: for each datum, modules are combined at each layer according to their normalized scores without requiring a task's ID (§3). The local component is also used for module expansion: a new module is instantiated if all the current modules flag their input as being locally out-of-distribution (§3.1). Further, new lower-level modules are first trained in a *projection phase*, to maximize the relatedness scores of subsequent modules (§3.2). This projects the output of low-level modules into the same representation space as the higher-level modules and ensures the compatibility between low- and high-level modules.

We explore the performance and versatility of our local structural approach which we call Local Module Composer (LMC) in a set of studies. First, we show that LMC reaches superior or comparable performance to existing modular and non-modular methods without requiring task IDs at test time using the Continual Transfer Learning (CTrL) benchmark, designed to evaluate transfer and forgetting in CL [73] (§4.1). Then, we demonstrate how LMC, relying on its projection phase, can solve out-of-distribution (OOD) tasks not seen during the continual training (§4.2). We also show it is possible to combine modules from independently trained models into a new model to solve tasks seen by each of the independent models without any finetuning (§4.3). Finally, we demonstrate the applicability of LMC in the meta-continual learning (meta-CL) setting, a setting that is task-agnostic by nature (§4.4). In meta-CL, LMC performs competitively when compared with challenging baselines such as experience replay [63] and achieves superior performance when tested on mixture tasks — tasks containing samples from several task-distributions seen during meta-training.

We highlight that by relying on a local (per-module) structural component LMC offers a modular CL approach that: i) does not require task IDs during test in the standard task incremental settings; ii) balances parameter sharing to yield strong CL performances compared to baselines that require access to the task ID; iii) in our experiments instantiates a sub-linear number of modules; iv) permits recombination of modules at test time enabling OOD generalization.

2 Background: Modular Continual Learning

Let $\mathcal{F}(x;\theta) : \mathcal{X} \to \mathcal{Y}$ be a learner parametrized with a set of parameters θ. In task incremental CL the learner is exposed to a sequence of tasks. Each task is composed of a training set D_t of (x, y)

pairs and a task identifier (ID) t [73, 36]. The goal is to learn an optimal θ^* that minimizes the loss $\mathcal{L}$ for all observed tasks:

$$\theta^* = \arg\min_{\theta} \sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim D_t}[\mathcal{L}(\mathcal{F}(x;\theta), y)]. \tag{1}$$

The parameter sharing trade-off between tasks can be addressed through different architectural design choices for $\mathcal{F}$. For example, $\mathcal{F}$ can be a monolithic network that shares parameters θ across all tasks. Most existing task incremental CL methods use a task-specific output head, requiring the task ID to select the one corresponding to the task at hand [36, 69, 1]. At the other end of the spectrum, a mixture of experts (MoE) learns an independent model for each task [2, 65, 2]. In this case, each expert trains task-specific parameters $\theta = \{\theta^{(t)}\}_{t=0}^{T}$.

To balance parameter sharing and transfer, modular methods organize their parameters in a series of modules $M = \{m_k^l\}$ with parameters $\theta = \{\theta_k^{(l)}\}$, where $\theta_k^{(l)}$ denotes the parameters of module k at layer l in $\mathcal{F}$. In general, a module can be any parametric function. In our experiments, unless otherwise stated, a module consists of a single convolutional layer followed by batch-norm, ReLU activation and a max-pooling operation. Modules can be composed conditioned on a sample, a batch of samples or a task. Let ψ denote a specific composition of modules that gives rise to a distinct prediction function, we make this dependence explicit: $\mathcal{F}(x;\theta,\psi)$. Importantly, in contrast to MoE and monolithic architectures, modules can be shared across tasks.

Veniat et al. [73] frames modular CL as finding an optimal layout $\psi^{(t)}$ for each task, where each layout selects a single module per layer per task (hard selection). In this case the set of layouts $\Psi = \{\psi^{(t)}\}$ grows with the number of tasks, while modules can be reused across different task-specific layouts:

$$\theta^*, \Psi^* = \arg\min_{\theta,\Psi} \sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim D_t}[\mathcal{L}(\mathcal{F}(x;\theta,\psi^{(t)}), y)]. \tag{2}$$

They design a method called MNTDP to search the exponentially large space of modular layouts, by only considering layouts resulting from adding a new module per layer to the best prior path (past task's path with the highest nearest neighbour accuracy on a new task) starting at the top layer. This solutions relies on task IDs to retrieve $\psi^{(t)}$ at test time.

Another way of composing modules uses dynamic routing [49, 64, 37, 51]. The module layout is generated by a structural function $\psi = s(x)$, hence different inputs take different routes through $\mathcal{F}$. It is standard to approximate the structural function using a neural network $\psi = s(x;\phi)$ with structural parameters ϕ. This framework has been applied to CL in [49] by learning a separate structural function per task $\psi^{(t)} = s(x;\phi^{(t)})$. The task IDs are used to retrieve the correct structural function:

$$\theta^*, \Phi^* = \arg\min_{\theta,\Phi} \sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim D_t}[\mathcal{L}(\mathcal{F}(x;\theta,s(x;\phi^{(t)})), y)], \tag{3}$$

where $\Phi = \{\phi^{(t)}\}$ is the set of structural parameters for all tasks.

The above methods require task IDs at both training and testing time. Next we introduce our modular CL approach that only relies on task IDs during training.

3 Local Module Composer (LMC)

We propose LMC, a modular approach where each module consists of a functional component $f(x;\theta_m^{(l)})$ and a structural component $s(x;\phi_m^{(l)})$, see Figure 1. The functional components are responsible for learning to solve the prediction task and are trained via the usual task loss $\mathcal{L}$ (e.g. cross-entropy loss for classification). The structural components are responsible for dynamic routing through $\mathcal{F}$ and are trained via the structural loss $\mathcal{L}^{(st.)}$, a loss computed locally at each module. We first describe both components including routing and then describe module expansion (§ 3.1) and how it impacts learning (§ 3.2).

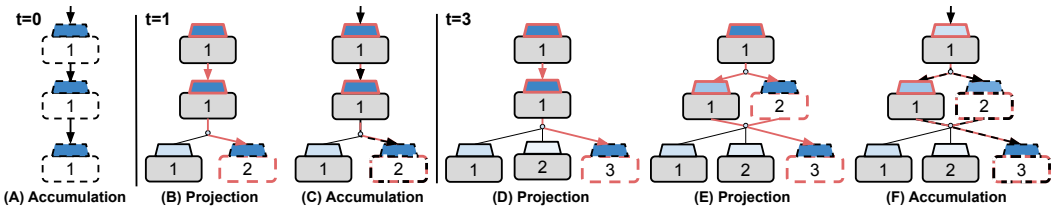


Figure 2: **Two-phase training**. Each module contains a functional (square) and structural (trapezoid) component, the attention strength is represented by the color intensity. Components with dashed contours are trained, solid contours represent fixed components, arrows show the gradient flow: black arrow – *functional* signal, pink – *structural* signal. **(A)** All components of the learner are trained on task 0. **(B)** Task 1 arrives, a new module is added at layer 1, which is first trained to project its output into the representation of the modules above via the *structural* signal (the *functional* signal is optional). No module addition is allowed during the projection phase. **(C)** Module addition is allowed again, both signals are used for training. **(D)** As task 3 arrives, a new module is added at layer 1 again, projection phase is triggered. **(E)** A new module is added at the layer 2, both new modules are now trained in the second projection phase. **(F)** Both new modules are trained using both signals.

Given an input sample $x = x^{(0)}$, the output $x^{(l)}$ of layer l is defined as the weighted sum of the functional outputs of all local modules and used as input to the subsequent layer $l+1$:

$$x^{(l)} = \sum_{m=1}^{|M^l|} w_m^{(l)} \cdot f(x^{(l-1)}; \theta_m^{(l)}). \tag{4}$$

Next, we compute module activation weights $w_m^{(l)}$ by normalizing the vector of local relatedness scores $\gamma^{(l)} \in \mathbb{R}^{|M^{(l)}|}$, each element of which is given by the negative structural loss:

$$\gamma_m^{(l)} = -\mathcal{L}^{(st.)}\Big(s\big[f(x^{(l-1)}; \theta_m^{(l)}); \phi_m^{(l)}\big]\Big), \tag{5}$$

$$w_m^{(l)} = softmax(\gamma^{(l)})_m. \tag{6}$$

Instead of using the $softmax$ function, it is possible perform hard selection taking the module with the highest score, or alternatively selecting top-k modules. In both cases, LMC stays end-to-end differentiable due to the local nature of structural learning. Note that in case of global structural objective, hard module selection would require applying tools for non-differentiable learning such as Expectation Maximization [37] or reinforcement learning based methods [64, 5].

The overall LMC objective consists of optimizing both functional and structural losses:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim D_t}\Big[\mathcal{L}\big(\mathcal{F}(x;\theta,\phi), y\big) + \sum_{l=0}^{L}\sum_{m=0}^{|M^l|} \mathcal{L}_m^{(st.)}\big(s[f(x^{(l-1)}; \theta_m^{(l)}); \phi_m^{(l)}]\big)\Big]. \tag{7}$$

As in [73], learning is performed w.r.t. only newly introduced modules to prevent forgetting.

Structural component. We use two possible instantiations of the structural component s and loss $\mathcal{L}_m^{(st.)}$. On the first one, s is an invertible neural network. Here we used the invertible architecture proposed by Dinh et al. [19]. As shown by Hocquet et al. [30], the structural objective can be defined as $\mathcal{L}_m^{(st.)}(x) = ||x||_2$. Intuitively, an invertible architecture prevents $\mathcal{L}_m^{(st.)}$ from collapsing to an all-zeros solution. In the second instantiation, s and f form an autoencoder and $\mathcal{L}_m^{(st.)}(x) = ||x^{(l-1)} - x||^2$ is the reconstruction error with respect to the module's input $x^{(l-1)}$. A similar idea was used in [2] for MoE training. Unless otherwise stated, modules in the feature extractor use the autoencoder as their structural component, while output heads use invertible s.

3.1 Expansion strategy

To acquire new knowledge, it is necessary to expand $\mathcal{F}$ as new tasks arrive. To this end, when existing modules detect outlier samples that exhibit large shift in the output space of their structural component, we fix the modules at this layer and add a new one. To detect such shifts we track the running statistics of the relatedness score γ for each module — mean μ and variance σ (see Figure 1). We then calculate a module-specific z-score as:

$$z_m = \frac{w_m - \mu_m}{\sigma_m}. \tag{8}$$

A sample is considered an outlier if the z-score z_m is larger than some predefined threshold z'. Note, that the expansion decision can be made on the per-sample or a per-batch basis. Unless otherwise stated, the decision was made on a per-batch basis in out experiments.

3.2 Training

Each module in LMC receives two types of learning signal: a structural signal resulting from minimizing $\mathcal{L}_m^{st.}$, and a functional signal resulting from minimizing the global functional loss $\mathcal{L}$. All structural components s are trained only with the structural signal that is calculated locally at each module. The training of functional components proceeds in two phases: projection and accumulation. Every new module addition triggers the projection phase. During this phase, the new module is trained to minimize the structural loss from all the layers above and no new-module addition is allowed. This procedure makes the representation of new modules compatible with subsequent modules and enables their composition. Intuitively, this procedure "encourages" already-learned modules to be reused preventing over-spawning of new modules. The functional signal is optional during projection (we kept it in all experiments unless otherwise stated). In the accumulation phase new module addition is allowed again and all modules are trained with both signals. The two-phase training is explained schematically in Figure 2 and implementation details are provided in §A.

4 Experiments

We now evaluate the performance, empirical capabilities, and properties of LMC in four different CL settings. First, in § 4.1 we study a standard task-incremental CL setting using the Continual Transfer Learning Benchmark (CTrL) [73]. Next, we explore the properties of LMC through other CL settings to modular approaches. In § 4.2 we evaluate the continual OOD generalization ability of our model. In § 4.3 we show the ability of LMC to combine modules form independently trained models. Last, we demonstrate LMC can perform well in the challenging Continual Meta-Learning setting in § 4.4.

4.1 Continual transfer learning (CTrL)

The CTrL benchmark was proposed to systematically evaluate properties of CL methods with a focus on modular architectures [73]. It consists of 5 streams of visual image classification tasks. The first stream $S^- = (t_1^+, t_2, t_3, t_4, t_1^-)$ consists of this sequence of 5 tasks, where the first and last task are the same, except the first has an order of magnitude more samples ("$^+$") than other tasks. This stream is designed to evaluate the *direct transfer* ability of models, i.e. a modular learner should be able to *reuse* the first task's modules for the last task. The $S^+ = (t_1^-, t_2, t_3, t_4, t_1^+)$ stream is similar to S^-, but the last task comes with more data then the first one. Here, the modular learner should be able to *update its knowledge*, i.e. performance on the first task should improve after learning the last task. In the $S^{in} = (t_1, t_2, t_3, t_4, t_1')$ stream the first task t_1 and the last t_1' tasks are similar, with a slight input distribution change (e.g. different background color). In the $S^{out} = (t_1^+, t_2, t_3, t_4, t_1'')$ stream the first task t_1 and the last task t_1'' differ in the amount of training data and the output distribution, i.e. the labels of the last task are randomly permuted. The *plasticity* stream $S^{pl} = (t_1, t_2, t_3, t_4, t_5)$ evaluates the ability to learn a stream of unrelated and potentially interfering tasks, i.e. transfer from unrelated tasks can harm performance (descriptive statistics for all datasets are in Appendix B.1).

We compare to several baselines. **Finetune**: trains a single model (larger model marked with **L**) for all tasks. **Experts**: trains a model per task. The next are strong recent modular CL methods. **MNTDP** [73]: a recent search-based module selection approach described in more detail in § 2. MNTDP requires the task ID to retrieve the previously found best structure for each test task.

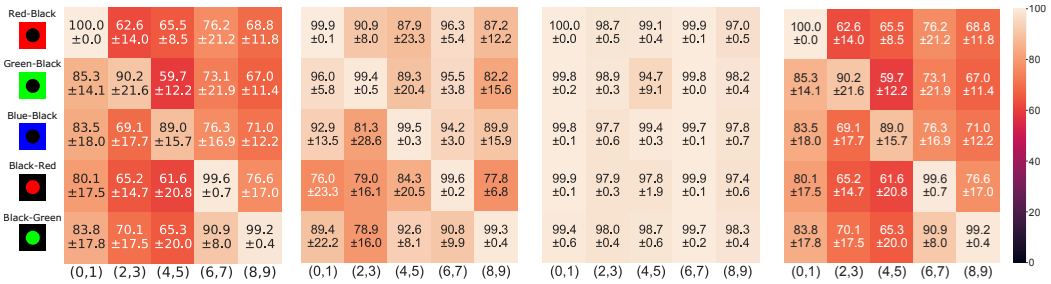
	S^-			S^+			S^{in}			S^{out}			S^{pl}		
MODEL	$\mathcal{A}$	$\mathcal{F}$	M	$\mathcal{A}$	$\mathcal{F}$	M	$\mathcal{A}$	$\mathcal{F}$	M	$\mathcal{A}$	$\mathcal{F}$	M	$\mathcal{A}$	$\mathcal{F}$	M
HAT[69]	54.5±1.3	-0.6±0.6	24*	51.1±2.3	-0.8±1.2	24*	48.0±1.4	-2.3±0.9	24*	54.4±2.4	-0.6±0.7	24*	56.0±0.9	-2.0±1.1	20*
EWC[36]	61.0±0.8	-5.9±1.2	24*	53.2±1.0	-5.8±1.2	24*	60.2±1.0	-4.6±1.1	24*	60.8±1.0	-6.0±1.0	24*	54.3±1.0	-2.7±1.4	20*
O-EWC[67]	59.1±3.3	-8.4±4.1	24*	52.6±1.9	-5.8±1.2	24*	59.7±1.0	-5.0±1.0	24*	61.3±2.4	-6.6±2.3	24*	50.8±1.9	-7.6±2.0	20*
ER[63, 15]	62.5±0.6	-1.4±0.7	4*	61.8±0.8	3.9±1.0	4*	57.7±2.3	3.2±1.2	4*	62.4±0.7	-0.9±0.5	4*	62.4±0.4	3.0±0.3	4*
EXPERTS	62.6±0.7	0.0	24	**62.8±0.9**	0.0	24	65.2±1.0	0.0	24	63.0±0.5	0.0	24	**64.2±0.6**	0.0	20
MNTDP[73]	65.8±0.6	0.0	14.6	62.6±1.0	0.0	21.5	66.4±0.9	0.0	15.0	66.2±0.8	0.0	13.7	63.4±0.5	0.0	15.7
SG-F[49]	63.6±1.1	0.0	14.7	60.5±0.5	0.0	18.5	63.9±1.5	0.0	19.2	63.6±1.8	0.0	14.3	62.8±1.4	0.0	16.7
LMC($\neg A$)	**66.6±1.5**	-0.0±0.1	15.3	60.1±2.7	-1.4±2.4	21.3	**69.5±1.0**	0.0±0.1	20.0	**66.7±2.2**	-0.1±0.1	15.5	61.6±4.8	-3.5±3.1	18.2
MNTDP(A)	44.5±3.0	-5.5±1.9	15.3	43.1±4.2	-9.2±5.4	20.7	27.3±2.5	-0.2±0.2	15.8	36.7±1.3	-4.8±2.5	14.8	25.4±2.4	-10.1±2.9	13.2
LMC(A)	**69.0±1.5**	-0.5±0.4	15.7	**62.2±4.5**	2.3±1.6	22.3	**68.5±1.7**	-0.1±0.1	19.7	**55.1±3.4**	-7.1±4.0	15.5	**60.5±4.9**	-4.4±4.7	18.2
LMC(A,H)	64.9±1.5	-0.2±0.2	16.2	55.8±2.5	-0.3±1.2	15.3	67.6±2.7	-0.8±1.0	21.5	54.2±3.6	-2.9±2.0	15.9	53.8±5.7	3.1±5.5	10.8
SG-F(A)	29.5±3.5	-35.3±4.0	14.3	20.4±4.4	-39.3±6.7	16	24.4±5.6	-38.7±4.0	18.7	30.5±4.5	-34.0±5.5	12.2	19.4±1.0	-41.8±1.6	15.5
ER(A,S)[63, 15]	60.4±0.7	1.0±1.3	4*	61.9±0.6	5.3±0.8	4*	57.7±1.0	5.0±3.5	4*	50.7±0.8	-0.7±1.8	4*	60.0±0.7	1.2±1.5	4*
FINETUNE	47.2±3.7	-14.6±3.8	4*	30.5±2.8	-28.4±2.9	4*	42.4±6.0	-22.5±6.5	4*	47.1±2.5	-14.0±2.7	4*	29.6±2.0	-28.6±2.1	4*
FINETUNE L	52.1±1.4	-15.7±1.7	24*	38.3±2.7	-24.0±2.8	24*	51.2±1.6	-13.3±1.8	24*	51.3±1.9	-17.3±2.1	24*	38.6±2.4	-24.1±2.2	20*

Table 1: **CTrL results:** we report accuracy (↑ $\mathcal{A}$), forgetting (↑ $\mathcal{F}$) with standard deviations calculated over 6 different runs. We report the mean number of modules (M) over these runs, where * marks methods with fixed capacity. The first block comprises a set of standard CL baselines including regularization and replay based methods. The second block are the modular methods, third – modular and replay based methods that are task ID agnostic (A), and the last block are the two finetuning baselines. (H) indicates hard module selection. (S) indicates single-head as detailed in the main text.

MNTDP(A): a task ID agnostic version of MNTDP we create which selects the path with the lowest entropy in the output distribution. **SG-F** [49]: Soft-gating with fixed modules, a modular method mentioned in § 2. It relies on a task-specific structural network that generates soft-gating vectors for each layer of the modular learner and fixes learned modules when new tasks arrive. We slightly adapted the original expansion strategy of SG-F in order to conform to our experimental setup, details are in § B.2. **SG-F**(A): a version of SG-F with a single structural net shared across tasks. **HAT**[69]: learns attention masks for activations that gate the gradients to prevent forgetting. The task ID is used to select task-specific attention mask for inference. The next are few standard CL methods. **EWC** [36]: trains a single model for all tasks while applying parameter-regularization to minimize forgetting. **O-EWC**:[67] online version of EWC that does not require storing a separate approximation of the Fisher information matrix per task. **ER** [15]: trains a single model while replaying samples from previously seen tasks. The size of the replay buffer corresponds to the memory size of the LMC assuming linear growth (i.e. LMC with 24 modules on a 5-task sequence). **ER**(A,S): task ID agnostic version of ER that uses a single output head to classify all classes from all tasks. **LMC**: We use several versions of LMC. **LMC**($\neg$A) a version of LMC that uses the task ID for output head selection (not module selection as MNTDP). **LMC**(A): the default version of LMC. It equips output heads with structural components and is therefore task ID agnostic at test time. **LMC**(A,H): a version of task ID agnostic LMC that performs hard module selection, i.e. taking the module with the highest relevance score per layer. Details on architecture and training are in § B.2.

Similar to [73], we use the following evaluation metrics: ($\mathcal{A}$) **average accuracy** on all seen tasks at the end of CL training; **Forgetting** ($\mathcal{F}$) — difference between accuracy at the end of the training and accuracy after learning the task (averaged across tasks) [46]; **Number of modules** (M) at the end of the continual training procedure. A formal definitions of the used metrics can be found in App. B.3.

Table 1 reports the results on the CTrL benchmark. Overall, modular methods tend to outperform ER and the regularization based methods (HAT and EWC). Among the modular methods, soft-gating SG-F(A,F) with a single controller shared among all the tasks performed the worst. This baseline showcases the problem of forgetting in the structural function or controller of dynamic routing methods such as [49]. A version of LMC performs the best on the S^+, S^{in} and S^{out} streams. Notably, LMC(A), which does not rely on task IDs, outperformed all other task ID agnostic methods such as MNTDP(A) and ER(A,S), and performed on par with task ID aware methods on all streams but S^{out}. Low performance is expected for methods relying on automated task inference on the S^{out} stream due to output distribution shift: i.e. at test time we notice that LMC correctly assigns samples



(a) EWC (b) MNTDP (c) LMC$_{(\neg A)}$ (d) LMC$_{(\neg A)}$ (no proj.)

Figure 3: Continual OOD-generalization: confusion magtricies show test accuracies on seen and unseen tasks. In each sub-figure x-axis shows the MNIST class-combination used to build the task, y-axis gives the foreground-background colors. Only diagonal tasks are learned continually. While non-modular EWC (a) and modular MNTDP (b) can prevent forgetting, LMC$_{(\neg A)}$ (c) is able to generalize to OOD tasks as well. LMC$_{(\neg A)}$ without projection phase performs poorly as shown in (d).

from the last task t_1'' to the first task's t_1^+ output head. However, the resulting classification accuracy is low because the labels were permuted for the last task. Interestingly, the task ID agnostic LMC$_{(A)}$ outperforms task ID aware LMC on the S^- and S^+ streams. Here, LMC$_{(A)}$ selects modules (and the output head) which were predominantly trained on the task that provided more training data (e.g. t_1^+ in S^- stream), hence fostering transfer between the first and the last tasks. In addition, LMC exhibits high variance on the S^+ and S^{pl} streams leading to it being outperformed by Experts and the MNTDP. This may be caused by the larger amount of trainable parameters compared to other models that induces higher variability. Finally, low performance of LMC$_{(H)}$ emphasizes the importance of soft modular attention for LMC. Additional results, including on a transfer metric [73], are in § B.4.

4.2 Compositional OOD generalization

This second study tests the ability of LMC to recombine modules for OOD generalization. We use a colored-MNIST dataset — a variation of the standard MNIST dataset of hand-written digits from 0 to 9 [34] in which digits are colorized. We design a simple sequence of tasks as follows. First, we define two high-level features: the foreground-background color combination (using the colors red, black, green, blue) and the class (0–9). Then, we create 5 non-overlapping tasks of two classes each: {`0-1, ..., 8-9`}. At training time the model is continually trained using a sequence of these tasks, however each task is only seen in one of five different foreground-background combination {`red-black, green-black, blue-black, black-red, black-green`}. At test time we measure the generalization ability to all unseen combinations of classes and colors.

In Figure 3 we present the confusion matrices for different learners when tested on all 25 combinations of colors and classes after it has been trained only on the 5 tasks on the diagonal. We compare the performance of LMC$_{(\neg A)}$ with EWC [36], MNTDP [73], and an ablated version of LMC without the projection phase. We observe that the OOD accuracy attained by LMC is significantly higher than EWC and MNTDP. Since the model trained with EWC is monolithic, the digit-background color combinations are entangled with the digits' shape for each task, which hinders OOD generalization. In contrast, modular approaches such as MNTDP and LMC learn a different module combination for each task. LMC's module selection is independent of task identifiers and each module is selected in a local manner based on its compatibility with the current input. Therefore, LMC can compose existing modules to adapt to new tasks based on their similarity with respect to previous tasks, even if the exact task has not been seen at training. In contrast, MNTDP must reuse the same module compositions learned during training based on task IDs. This forces MNTDP to use modules that were trained using a different color combination, and results in e.g. a 24% accuracy drop with respect to LMC on `[0,1]` when the foreground and background colors are inverted w.r.t. the seen combination.

In Figure 3d we report results for LMC without applying the projection phase. The projection phase adapts the representation of newly introduced modules to match the distribution expected by the subsequent modules. As expected, we found that skipping it severely degrades performance. This result validates the usefulness of the projection phase to achieve an efficient local module selection.

In Appendix D we plot the average module selection for all 25 test tasks, showing how modules are reused for the OOD tasks.

4.3 Combining modular learners

In earlier sections we show cross-task reusability of modules, here we test cross-model reusability. We motivate the practical importance of this kind of reusability with an example: a privacy preserving training might be required for LMC1 and LMC2, trained on the premises of customers 1 and 2, after which their modules can be combined in a single central entity — LMC3, located on premises of the cloud service provider. LMC3 is required to perform tasks seen by both independent LMCs and so might combine knowledge from both LMC1/2 to outperform each.

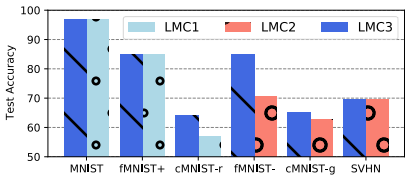


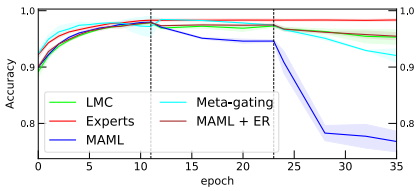
Figure 4: Performance of combining independently trained LMC1 and LMC2 into LMC3.

In Fig. 4 we test the ability of LMC to preserve and transfer knowledge in such setting. To this end we design the following tasks: fMNIST+ and fMNIST-. Both are sampled from the fashion-MNIST dataset [78] but the latter comes with much less training data. cMNIST-**r**/cMNIST-**g** are variants of the colored-MNIST dataset where the background of 95% of the training samples colored in **r**ed/**g**reen and 5% in **g**reen/**r**ed, while the test color proportions are 50%/50%. We trained LMC1 continually on MNIST, fMNIST+, and cMNIST-r tasks. On the other hand, LMC2 was trained on fMNIST-, cMNIST-g, and SVHN. We combined the modules of both LMCs layer wise to obtain LMC3.

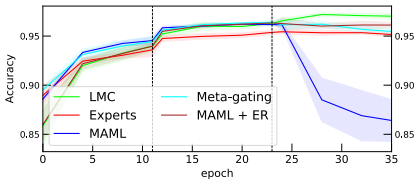
We observed positive transfer for both cMNIST and fMNIST- tasks with LMC3. We found that LMC3 selects different modules originating from different LMCs conditioned on test samples with different background colors — LMC1's modules were specialized on red background while LMC2's on green (selected paths presented in Appendix C). Notably, cross-model reusability without fine-tuning is novel to LMC and is attributed to the local nature of the structural component. Using a global structural component as in [49] would first require tuning a structural component specifically from LMC3. In case of task-specific routing [73], additional fine-tuning would be needed to discover task-specific paths through the consolidated model. In neither case privacy would be preserved.

4.4 Continual meta-learning

In this setting a learner needs to leverage the knowledge obtained via meta-training on a variety of tasks sampled from a task distribution (i.e. environment), while this distribution might change [33, 13, 28]. As discussed in [33], modularity can be especially useful in meta-learning with heterogeneous task distributions, where a single set of parameters learned by MAML [20] may not be sufficient to guarantee the model's ability to adapt to any new task within a few steps of gradient descent.



(a) Meta-test accuracies



(b) Mixtures of tasks meta-test accuracy

Figure 5: Results on the OMF sequence. Vertical dashed lines mark the environment switches.

We construct two environment sequences. In the first one, the learner is meta-trained on a sequence of Omniglot [40], MNIST [41] and fashion-MNIST [78] datasets — the OMF sequence. We evaluate models in two settings. In the first setting, we report the average meta-test test accuracy over the environments seen so far. In the second setting, models are evaluated on environment mixtures, i.e. each test task is composed of samples from the environments learned so-far. A possible mixture task could contain classes 1 and 2 sampled from Omniglot, 2 and 3 from MNIST and class 5 sampled from the fashion-MNIST environment. See Appendix E.3 for additional experiments on the *mini*Imagenet dataset [62].

We compare to the following baselines. **MAML**: non-modular MAML [20]. **MAML + ER**: MAML with experience replay. **Meta-gating** tries to learn the relevance scores $\gamma^{(l)}$, used for gating, directly in the inner-loop. This is similar to MetaCoG [28] applied to modules' outputs or soft-layer ordering used in [49]; **Experts**: an upper bound where separate monolithic model is trained for each environment and selected using the environment ID. **LMC** for meta-CL with slightly adopted expansion strategy. This and other details are in Appendix E. All experiments are performed in a 5-way 5-shot setting.

Results on OMF are shown in Figure 5. These results suggest that LMC can successfully avoid catastrophic forgetting achieving final average accuracy comparable to the experience replay (MAML + ER) baseline. This can be mainly attributed to the fact that learned modules at a layer are frozen every time a new module is added — modules are not updated on new tasks if these tasks have triggered module addition and thus were recognized as outliers at a given layer.

Results with mixtures of tasks are in Figure 5b. We find that LMC attains better performance than the other baselines. The performance improvement is due to LMC assigning modules on a per-sample basis in each forward pass, effectively guiding the inner- and outer-loop gradients of the meta-training procedure to the modules with higher activation on the current input. This contrasts the selection mechanism proposed by [33] for MoE, where estimating the probability of each expert involves computation of the inner-loss which can be performed only on a per-task level. While replay slightly underperforms LMC on the mixture task, MAML is unable to reach high accuracy due to catastrophic forgetting. Regarding the experts baseline, each expert only specializes on one of the three environments, which results in low performance on the mixture-task.

5 Related work

Modularity in neural networks has been studied in the context of scalability [9], and more recently as a way to achieve compositionality and systematic generalization [6, 37, 14, 8, 23, 18] as well as for multi-task learning [51, 64]. Another line of work has studied modularity as a way to learn independent mechanisms [66, 59, 23]. This idea was crystallized by [58], who proposed a way to learn a set of causal independent mechanisms as MoE. Building up on this ideas, others showed evidence of compositional OOD generalization [47]. Recently, [52] argued for a more wholistic view on CL including OOD generalization as an important desiderata. **Continual learning** methods typically address the problem of forgetting through parameter regularization [36, 55, 79], replay [70, 61, 32, 63, 3, 57, 44, 11, 77] or dynamic architectures (and MoEs) [65, 69, 42, 67, 43, 33]. Our work falls under the umbrella of the latter and shares its advantage of having the capacity to adapt to a large number of related tasks. Our focus is on improving **modular CL** approaches, which despite their advantages have only recently been studied in the CL literature [49, 73]. The main difference with our work is that we use a local composition mechanism as opposed to a global one. We discuss in more details this difference (§2) and also compare to these methods (§4.1). **Continual-meta learning** focuses on fast learning and remembering [21, 28, 27, 33], often emphasising the online performance on OOD tasks [13]. As argued by Jerfel et al. [33] modularity can be useful in this setting to minimize interference between tasks. They proposed a way to train a MoE model, with each expert focusing on a cluster of tasks leveraging Bayesian nonparametrics. LMC aims at decomposing knowledge into layer-wise composable modules further reducing modular granularity.

6 Conclusion

We develop LMC, a method to learn and compose a series of modules on a continual stream of tasks fulfilling some of the basic desiderata of modular CL such as module specialization, avoidance of collapse, and sublinear growth. In LMC, structural information is learned and stored locally to each module. It is the locality of structural component that enables generalization to related but unseen tasks as well as combining different LMCs without fine-tuning. Future work could focus on achieving more efficient sub-linear model growth through OOD generalization and reusability of modules. Additionally, while the benefits of modularity for CL are well understood, the implications of the CL regime on modularity and compositionality have not been studied extensively. It is possible that providing knowledge to the learner in incremental chunks results in the implicit supervision needed to better disentangle it into specialized and composable modules. Another promising directions is removing the need for task boundaries during training and developing more robust architectures for the local structural component (related discussions are in §A.3, and limitations in §F).

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. *CoRR*, abs/1711.09601, 2017.

[2] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3366–3375, 2017.

[3] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019.

[4] Mohammed Amer and Tomás Maul. A review of modularization techniques in artificial neural networks. *Artificial Intelligence Review*, 52(1):527–561, 2019.

[5] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*, 2016.

[6] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

[7] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

[8] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*, 2018.

[9] Dana H Ballard. Modular learning in neural networks. In *AAAI*, pages 279–284, 1987.

[10] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

[11] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with stacked quantization module. *ArXiv*, abs/1911.08019, 2019.

[12] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexande Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. In *arXiv preprint arXiv:2003.05856* Caccia et al. [13].

[13] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexande Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. *arXiv preprint arXiv:2003.05856*, 2020.

[14] Michael Chang, Abhishek Gupta, Sergey Levine, and Thomas L Griffiths. Automatically composing representation transformations as a means for generalization. In *International Conference on Learning Representations*, 2018.

[15] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019.

[16] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[17] Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. Modularity improves out-of-domain instruction following. *arXiv preprint arXiv:2010.12764*, 2020.

[18] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=7uVcpu-gMD.

[19] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[21] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *International Conference on Machine Learning*, pages 1920–1930. PMLR, 2019.

[22] Robert M French. Pseudo-recurrent connectionist networks: An approach to the'sensitivity-stability'dilemma. *Connection Science*, 9(4):353–380, 1997.

[23] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

[24] Anirudh Goyal, Aniket Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. *arXiv preprint arXiv:2103.01197*, 2021.

[25] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

[26] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. *arXiv preprint arXiv:2007.01434*, 2020.

[27] James Harrison, Apoorva Sharma, Chelsea Finn, and Marco Pavone. Continuous meta-learning without tasks. *arXiv preprint arXiv:1912.08866*, 2019.

[28] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu. Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201*, 2019.

[29] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[30] Guillaume Hocquet, Olivier Bichler, and Damien Querlioz. Ova-inn: Continual learning with invertible neural networks. *arXiv preprint arXiv:2006.13772*, 2020.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[32] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI conference on artificial intelligence*, 2018.

[33] Ghassen Jerfel, Erin Grant, Tom Griffiths, and Katherine A Heller. Reconciling meta-learning and continual learning with online mixtures of tasks. In *Advances in Neural Information Processing Systems*, pages 9122–9133, 2019.

[34] Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. Learning not to learn: Training deep neural networks with biased data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9012–9020, 2019.

[35] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[36] 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.

[37] Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. *Advances in Neural Information Processing Systems*, 31:2408–2418, 2018.

[38] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[39] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[40] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[41] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[42] Jeongtae Lee, Jaehong Yoon, Eunho Yang, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *CoRR*, abs/1708.01547, 2017.

[43] Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning. *arXiv preprint arXiv:2001.00689*, 2020.

[44] Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Jean-François Goudou, and David Filliat. Generative Models from the perspective of Continual Learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.

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

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

[47] Kanika Madan, Nan Rosemary Ke, Anirudh Goyal, Bernhard Schölkopf, and Yoshua Bengio. Fast and slow learning of recurrent independent mechanisms. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Lc28QAB4ypz.

[48] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[49] Jorge A Mendez and Eric Eaton. Lifelong learning of compositional structures. *arXiv preprint arXiv:2007.07732*, 2020.

[50] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

[51] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *arXiv preprint arXiv:1711.00108*, 2017.

[52] Martin Mundt, Yong Won Hong, Iuliia Pliushch, and Visvanathan Ramesh. A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning. *arXiv preprint arXiv:2009.01797*, 2020.

[53] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

[54] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[55] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[56] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[57] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11321–11329, 2019.

[58] Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pages 4036–4044. PMLR, 2018.

[59] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[60] Mohammad Pezeshki, Sékou-Oumar Kaba, Yoshua Bengio, Aaron Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. *arXiv preprint arXiv:2011.09468*, 2020.

[61] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[62] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

[63] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, 2019.

[64] Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

[65] 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.

[66] B Schölkopf, D Janzing, J Peters, E Sgouritsa, K Zhang, and J Mooij. On causal and anticausal learning. In *29th International Conference on Machine Learning (ICML 2012)*, pages 1255–1262. International Machine Learning Society, 2012.

[67] 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. *arXiv preprint arXiv:1805.06370*, 2018.

[68] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *International Conference on Machine Learning (ICML)*, 2018.

[69] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018.

[70] Andrea Soltoggio. Short-term plasticity as cause–effect hypothesis testing in distal reward learning. *Biological cybernetics*, 109(1):75–94, 2015.

[71] Olaf Sporns and Richard F Betzel. Modular brain networks. *Annual review of psychology*, 67:613–640, 2016.

[72] Saul Sternberg. Modular processes in mind and brain. *Cognitive neuropsychology*, 28(3-4):156–208, 2011.

[73] Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

[74] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.

[75] Ziyu Wang, Bin Dai, David Wipf, and Jun Zhu. Further analysis of outlier detection with deep generative models. *arXiv preprint arXiv:2010.13064*, 2020.

[76] James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5):1229–1262, 2017.

[77] Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. *Advances in Neural Information Processing Systems*, 31:5962–5972, 2018.

[78] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[79] Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task agnostic continual learning using online variational bayes, 2018.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   Yes. In the last three paragraphs of the introduction we provide references to the different sections that justify the different claims.
   (b) Did you describe the limitations of your work?
   Yes, see §F.
   (c) Did you discuss any potential negative societal impacts of your work?
   Yes, we include it here. In this work, we presented a modular architecture for continual learning. Achieving continual learning can have beneficial impact in form of reduction of computation cost of training of large models. Additionally, modularity can potentially further improve computational cost since for a given task only a subset of the modules can active. We do not foresee any direct negative societal impact derived from this work, beyond the potential negative societal impacts of ML in general. We also include a discussion on a broader societal impact in § G.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them?
   Yes.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results?
   Not applicable
   (b) Did you include complete proofs of all theoretical results?
   Not applicable
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)?
   Yes. The code can be found under: `https://anonymous.4open.science/r/compositionality-D695/README.md`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)?
   Yes, we discuss this in sections A,B,C,D,E.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?
   Yes, we report the standard deviation after multiple runs, see Table 1.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)?
   Most of the experiments were run on an internal cluster of NVIDIA Quadro RTX 8000 GPUs.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators?
   Yes.
   (b) Did you mention the license of the assets?
   Yes.
   (c) Did you include any new assets either in the supplemental material or as a URL?
   No.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating?
   We use publicly available datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? **Not applicable.**
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable?
   Not applicable.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable?
Not applicable.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation?
Not applicable.