Continual Learning via Local Module Composition

Oleksiy Ostapenko[12] **Pau Rodríguez**[3] **Massimo Caccia**[123] **Laurent Charlin**[145]
[1]Mila - Quebec AI Institute, [2]Université de Montréal, [3]ServiceNow, [4]HEC Montréal,
[5]Canada CIFAR AI Chair

Abstract

Modularity is a compelling solution to continual learning (CL), the problem of modeling 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 provided a local structural component that estimates a module's relevance to the input. Dynamic module composition is performed layer-wise based on local relevance scores. We demonstrate that agnosticity to task identities (IDs) arises from (local) structural learning that is module-specific as opposed to the task- and/or model-specific as in previous works, making LMC applicable to more CL settings compared to previous works. In addition, LMC also tracks statistics about the input distribution and adds new modules when outlier samples are detected. In the 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 that the locality of structural learning allows LMC to interpolate to related but unseen tasks (OOD), as well as to compose modular networks trained independently on different task sequences into a third modular network without any fine-tuning. Finally, in search for limitations of LMC we study it on more challenging sequences of 30 and 100 tasks, demonstrating that local module selection becomes much more challenging in presence of a large number of candidate modules. In this setting best performing LMC spawns much fewer modules compared to an oracle based baseline, however it reaches a lower overall accuracy. The codebase is available under `https://github.com/oleksost/LMC`.

1 Introduction

The goal of continual learning (CL) is to learn efficiently from a non-stationary stream of tasks without (catastrophically) forgetting previous tasks [62]. CL is often modeled as a trade-off between knowledge retention (stability) and knowledge expansion (plasticity) [26, 64]. 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 [46, 57]. Conversely, learning a separate model per task eliminates forgetting but minimizes transfer and data efficiency [2, 41].

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, 47, 27]. In principle, a modular learner capable of composing modules in meaningful structures can provide additional benefits including **(i)** computational gains due to only executing modules that are relevant to a task [47, 4]; **(ii)** memory gains due to instantiating a sub-linear number of modules w.r.t. the number of tasks; **(iii)** systematic [8] and out-of-distribution (OOD) generalization [18] through knowledge recombination; and **(iv)** biological plausibility [91, 90, 96].

corresponding author: oleksiy.ostapenko@t-online.de

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Designing modular methods for CL comes with two main challenges. The first is *how and when to add new modules* to ensure sufficient plasticity to learn new tasks. Existing modular methods use greedy search variants, expanding the model when it improves validation performance [92, 63]. The 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 the form of an optimal module layout [92] or as a model- and task-specific controller network that generates modular layouts [63]. Unfortunately, in many realistic CL scenarios task identities are unavailable at test time [23, 35, 14]. 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 shallow modules (i.e. closer to the input) are first trained in a *projection phase* to maximize the relatedness scores of subsequent, deeper, modules (§3.2). This process projects the output of new modules into the representation space expected by the subsequent modules and ensures the compatibility between low- and high-level modules.

In a set of studies, we explore the performance and versatility of our local structural approach, which we call Local Module Composer (LMC). 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 [92] (§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, an analysis of longer task sequences (30 and 100 tasks) reveals that LMC tends to spawn much fewer modules to reach good performance than the fully task-aware MNTDP [92] counterpart. However, LMC reaches slightly lower accuracy on longer sequences than MNTDP, which highlights the difficulty of automatic task-ID agnostic module selection in the presence of a large number of candidate modules. In Appendix F we demonstrate the applicability of LMC in the meta-continual learning (meta-CL) setting, a task-agnostic setting by nature.

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 as well as (v) the ability to combine independently trained models in a third model without fine-tuning. Notably, the OOD generalization is only possible if the agent is task-agnostic in the module selection process, since OOD tasks were not observed at training, the learner has to interpolate between the learned tasks, and a (categorial) task ID is of no use.

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 [92, 46]. The goal is to learn an optimal θ^* that minimizes the loss $\mathcal{L}$ for all observed tasks:

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

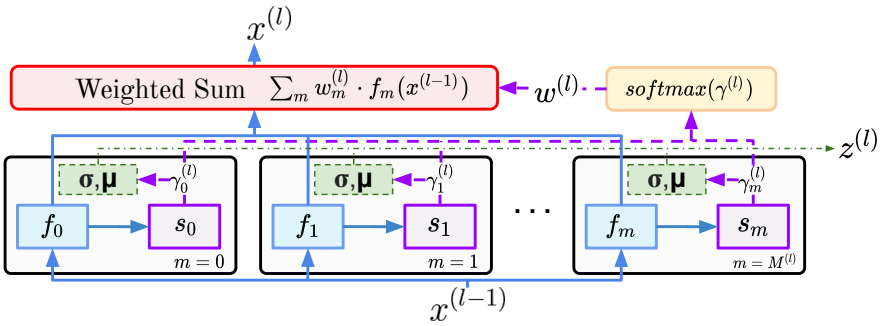


Figure 1: **Modular Layer Scheme**. Each black rectangle is a module. Inside each module, the functional component f_m receives the input $x^{(l-1)}$ and feeds its output to the structural component s. The output of the structural component is used to calculate the importance score $\gamma_m^{(l)}$ using 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.

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 output head corresponding to the task at hand [46, 87, 1].

At the other end of the spectrum are the expert based solutions that learn an independent model, a.k.a. expert, for each task [2, 83]. In this case, each expert trains task-specific parameters $\theta = \{\theta^{(t)}\}_{t=1}^{T}$.

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, sharing modules across tasks should lead to desirable transfer properties.

Veniat et al. [92] 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):

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

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 resulting in sub-linear growth pattern. 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 neighbor accuracy on a new task) starting at the top layer. This solution relies on task IDs to retrieve $\psi^{(t)}$ at test time.

Another way of composing modules uses dynamic routing [63, 82, 47, 65]. 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 [63] 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.

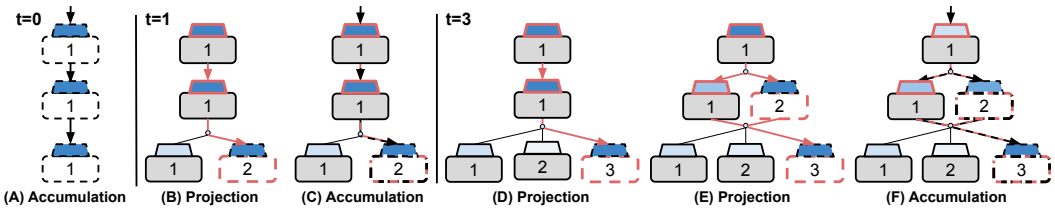


Figure 2: **Two-phase training**. Each module contains a functional (rectangle) and structural (trapezoid) component. Their color intensity denotes the strength of their activation. Components with dashed contours are trained, solid contours represent fixed components, arrows show the gradient flow: black arrow — *functional* signal, pink — *structural* signal. **(A)** All modules 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.

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 receive the corresponding functional output as their input (see Figure 1) and are responsible for dynamic routing through $\mathcal{F}$. Structural parameters ϕ are trained using a structural loss $\mathcal{L}^{(st.)}$ computed locally at each module.

Intuitively, the structural component of a module should serve as a density estimator of the outputs of the functional component. The module's contribution to the layer's output is proportional to the likelihood of the input sample under the estimated density. In our instantiation, the structural component produces a relatedness score: a lower score for inputs that are more likely to belong to the distribution on which a given module was trained, and a higher score for inputs that are out-of-distribution for the given module. Hence, the likelihood of the input sample is approximated by the negative relatedness score.

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

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

The functional output of the network is equal to the output of the final layer: $\mathcal{F}(x;\theta,\phi) = x^{(L)}$. In the last layer $\mathcal{F}$ implements a single output-head per task. At training time, the task ID is available and we update only the output-head corresponding to the currently learned task. At test time, the task ID is not available and we select the output head with the highest activation weight $w_m^{(L)}$, i.e., the last layer performs hard module selection.

The module activation weights $w_m^{(l)}$ are computed by normalizing the vector of local relatedness scores $\gamma^{(l)} \in \mathbb{R}^{|M^{(l)}|}$. Each element of $\gamma^{(l)}$ is obtained from the negative structural loss which approximates the likelihood of each module:

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

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

Modules with lower structural loss get higher activation weights. Note that in practice, it can be useful to bias the module selection towards the expected module selection in a batch, assuming that samples within a batch are likely to belong to the same task. We discuss this point further in § A.1.

Instead of using the softmax function, it is possible perform hard selection taking the module with the highest score [82], or alternatively selecting top-k modules [88]. In both cases, LMC's structural parameters stay differentiable due to the local nature of structural learning. Note that in the case of global structural objective, hard module selection would require applying tools for non-differentiable learning such as Expectation Maximization [47] or reinforcement-learning based methods [82, 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 [92], learning is performed w.r.t. only newly introduced modules to prevent forgetting.

Structural component. We test two instantiations of the structural component s and loss $\mathcal{L}_m^{(st.)}$. In the first one, s is an invertible neural network [80]. Here we use the invertible architecture proposed by Dinh et al. [21]. As shown by Hocquet et al. [37], for this invertible architecture 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 $x^{(l-1)}$. Aljundi et al. [2] used a similar idea was for selecting the most relevant expert network conditioned on a task. Unless stated otherwise, modules in the feature extractor use the autoencoder as their structural component, while output heads use invertible s — these combinations worked well in practice.

3.1 Expansion strategy

It is necessary to expand $\mathcal{F}$ as new tasks arrive to acquire new knowledge. A new module is added to a layer when all modules in this layer detect an outlier input. To this end, we track the running statistics of the relatedness score γ for each module — mean μ and variance σ (see Figure 1), and calculate a z-score for each sample in the batch and each module at a layer:

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

An input is considered an outlier if its z-score is larger than a predefined threshold z' (see Appendix B.6 for an ablation study of z' values). The expansion decision can be made on the per-sample (i.e., if an outlier sample is detected) or a per-batch basis (i.e., z is averaged over the mini-batch). Unless stated otherwise, in our experiments, the decision was made on a per-batch basis. Additionally, at training the parameters of existing modules are fixed once the task changes. If during a forward pass through $\mathcal{F}$ module addition is triggered at multiple layers, we start adding modules at the layer closest to the input.

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 to each module.

The training of functional components proceeds in two phases: projection and accumulation. Whenever the expansion strategy triggers the addition of a new module (i.e., $z_m > z' \; \forall m \in \{0, \ldots, |M^{(l)}|\}$), starting with layers closest to the input, LMC initiates 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

	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[87]	63.7±0.7	-1.3±0.6	24*	61.4±0.5	-0.2±0.2	24*	50.1±0.8	0.0±0.1	24*	61.9±1.3	-3.2±1.3	24*	61.2±0.7	-0.1±0.2	20*
EWC[46]	62.7±0.7	-3.6±0.9	24*	53.4±1.8	-2.3±0.4	24*	56.3±2.5	-9.1±3.3	24*	62.5±0.9	-3.6±0.9	24*	54.2±3.1	-4.2±2.7	20*
O-EWC[85]	62.0±0.7	-3.2±0.7	24*	54.6±0.7	-1.3±1.0	24*	54.2±3.1	-10.8±3.1	24*	62.4±0.6	-3.0±0.9	24*	52.3±1.4	-5.7±1.3	20*
ER[81, 16]	60.6±0.7	-2.1±0.9	4*	63.0±0.6	3.8±0.8	4*	63.8±1.4	-1.9±0.6	4*	60.7±1.0	-1.5±0.5	4*	60.5±1.0	0.5±0.9	4*
EXPERTS	62.7±0.9	0.0	24	**63.2±0.8**	0.0	24	63.1±0.7	0.0	24	63.1±0.7	0.0	24	**63.9±0.5**	0.0	20
MNTDP[92]	66.3±0.8	0.0	13.7	62.6±0.7	0.0	21.0	67.9±0.9	0.0	16.0	65.8±0.9	0.0	15.0	64.0±0.2	0.0	17.2
SG-F[63]	63.6±1.5	0.0	14.7	61.5±0.6	0.0	20.8	65.5±1.8	0.0	17.5	64.1±1.3	0.0	16.2	62.0±1.3	0.0	16.0
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)	41.9±2.5	-2.8±0.6	14.8	43.2±1.3	-10.8±2.0	20.7	32.7±13.6	-15.2±13.2	17.2	37.9±2.7	-5.8±3.5	13.3	35.1±3.6	-16.4±4.6	15.8
LMC(A)	**67.2±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	**63.5±1.9**	-1.0±1.5	19.0
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	16.0	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)[81, 16]	60.4±1.0	-0.5±0.7	4*	**65.3±0.9**	6.0±1.0	4*	58.8±3.2	-4.2±3.7	4*	47.6±1.5	-7.6±1.6	4*	58.6±1.3	-1.2±1.5	4*
FINETUNE	47.5±1.5	-14.9±1.4	4*	31.4±3.7	-29.3±3.8	4*	39.7±5.0	-23.9±5.7	4*	45.4±4.0	-15.5±3.7	4*	29.1±3.1	- 29.2±3.2	4*
FINETUNE L	52.1±1.4	-15.7±1.7	24*	38.2±3.2	-25.8±3.3	24*	49.3±2.0	-18.4±2.0	24*	49.3±2.1	-18.4±2.0	24*	37.1±2.1	-26.0±2.2	20*

Table 1: **CTrL results:** we report accuracy ($\uparrow \mathcal{A}$), forgetting ($\uparrow \mathcal{F}$) 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.

compatible with subsequent modules and enables their composition. This procedure "encourages" already-learned modules to be reused, preventing over-spawning 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 non-frozen modules are trained with both signals. The functional components of new modules still receive a signal from the structural components of modules above. The two-phase training is explained schematically in Figure 2 and implementation details are provided in Appendix 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 (task-ID agnostic and aware) using the Continual Transfer Learning Benchmark (CTrL) [92]. Next, we explore the properties of LMC through other CL settings. In § 4.2 we evaluate the continual OOD generalization ability of the proposed LMC. In § 4.3 we show the ability of LMC to combine modules form independently trained models. In Appendix F, we evaluate LMC in the Continual Meta-Learning setting.

4.1 Continual transfer learning using the CTrL benchmark

The CTrL benchmark was proposed to systematically evaluate properties of CL methods with a focus on modular architectures [92]. It consists of 5 streams of visual image classification tasks. The first stream $S^- = (t_1^+, t_2, t_3, t_4, t_5, t_1)$ consists of a sequence of 6 tasks, where the first and last task are the same except the first has an order of magnitude more training 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 $S^+ = (t_1, t_2, t_3, t_4, t_5, t_1^+)$ stream is similar to S^-, but now the last task comes with more data than the other tasks (including 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 $S^{in} = (t_1, t_2, t_3, t_4, t_5, t_1')$ stream the first t_1 and the last t_1' tasks are similar, with a slight input distribution change (e.g. different background color). In the $S^{out} = (t_1^+, t_2, t_3, t_4, t_5, t_1'')$ 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 (wider model marked with **L**) for all tasks. **Experts**: trains a model per task. We also compare with the several recently proposed modular CL baselines, which achieve competitive results in CTrL and require task IDs at test time. **MNTDP** [92]: 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. **MNTDP**$_{(A)}$: a task ID agnostic version of MNTDP we created, which selects the path with the lowest entropy in the output distribution. **SG-F** [63]: 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 Appendix B.2. **SG-F**$_{(A)}$: a version of SG-F with a single structural network shared across tasks. **HAT**[87]: learns attention masks for activations that gate the gradients to prevent forgetting. The task ID is used to select a task-specific attention mask for inference.

We also compare to several standard CL methods. **EWC** [46]: trains a single model for all tasks while applying parameter-regularization to minimize forgetting. **O-EWC**:[85] online version of EWC that does not require storing a separate approximation of the Fisher information matrix per task. **ER** [16]: 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 the worse case linear growth pattern (i.e., LMC with 24 modules on a 6-task sequence). **ER**$_{(A,S)}$: task ID agnostic version of ER that uses a single output head to classify all classes from all tasks: i.e. after learning stream S^- the output head has 50 output neurons and the output classes of the last task are considered the same as the ones of the first task.

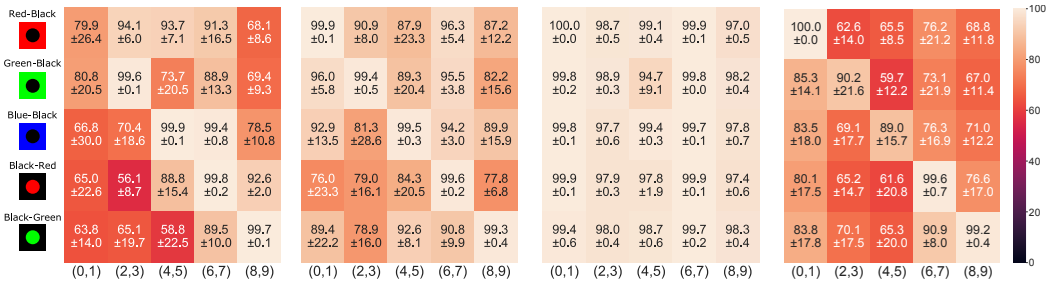
We use several versions of LMC. **LMC**$_{(\neg A)}$ a version of LMC that uses the task ID for output head selection (not module selection as MNTDP). **LMC**$_{(A)}$: the default version of LMC. It equips output heads with structural components and is therefore task ID agnostic at test time. **LMC**$_{(A,H)}$: a version of task ID agnostic LMC that performs hard module selection, i.e., taking the module with the highest relevance score per layer. All methods use the same architecture (described in Appendix A.4) together with the Adam [44] optimizer. HAT is the only method that uses SGD.

Similar to Veniat et al. [92], 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 [60]; **Number of modules** (M) at the end of the continual training procedure. Formal definitions of all metrics are in Appendix B.3.

Table 1 reports performance using 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 global structural component (a.k.a. controller) of dynamic routing methods such as the one proposed by Mendez and Eaton [63]. 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 at test time, outperformed all other task ID agnostic methods such as MNTDP$_{(A)}$ and ER$_{(A,S)}$ on all streams but S^+, and always performed on par with task ID aware methods. On the S^{out} stream low performance is expected for task-ID agnostic methods due to output distribution shift: i.e., at test time we notice that LMC correctly assigns samples 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 of the last task are randomly permuted in this stream.

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 transferring knowledge between the first and the last tasks. In contrast, LMC$_{(\neg A)}$ when tested on the last task t_1 is forced to select the output head belonging to this task, which was trained on less data than the output head of t_1^+ task, leading to lower accuracy. In addition, we observed that versions of LMC often exhibit high variance (e.g. see S^+, S^{out} and S^{pl} streams). This may be caused by the larger amount of trainable parameters compared to other models and relatively small amount of training data. Finally, low performance of



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

Figure 3: Continual OOD-generalization: matrices show test accuracy on seen and unseen tasks (on- and off-diagonal tasks respectively). 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).

LMC$_{(A,H)}$ emphasizes the importance of soft modular attention for LMC. Additional results, including a transfer metric [92], are in Appendix B.4.

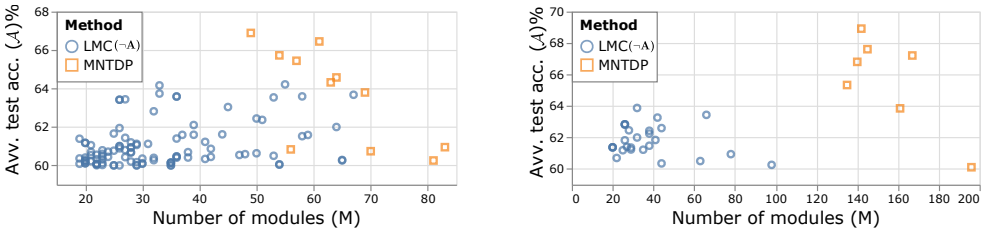
4.2 Compositional OOD generalization

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 [43] 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 five non-overlapping tasks of two (digit) 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 combinations {`red-black, green-black, blue-black, black-red, black-green`}. At test time we measure the generalization ability to seen and unseen combinations of classes and colors.

In Figure 3 we present the accuracy 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 [46], MNTDP [92], 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, hindering OOD generalization. In contrast, modular approaches such as MNTDP and LMC learn a different module combination for each task. In contrast to MNTDP, LMC's module selection does not rely on task identifiers and each module is selected in a local manner based on its compatibility with the current input. This allows LMC to interpolate between previously seen tasks being able to dynamically compose existing modules to adapt to tasks that have not been seen at training. Because MNTDP's module selection relies on a database of task-specific structures found to be optimal for the corresponding task at training, this method must reuse the predefined module compositions 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 E we plot the average module selection for all 25 test tasks, showing how modules are reused for the OOD tasks.



(a) 30 tasks sequence S^{long30}

(b) 100 tasks sequence S^{long}

Figure 5: Results on S^{long} and S^{long30} sequences for different hyperparameter values (we select only runs with reasonably good performance, i.e. $\mathcal{A}$>60%), same plots plots for all conducted runs can be found in Appendix C).

4.3 Combining modular learners

In earlier sections we show cross-task reusability of modules, here we test the cross-model reusability. We motivate the practical importance of this kind of reusability with a federated learning 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 but can not be finetuned as it has no access to the original training data distributions.

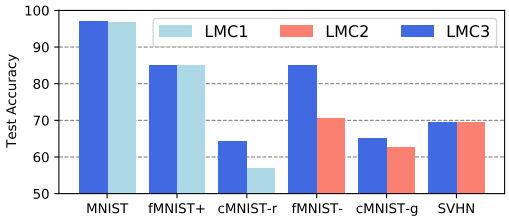


Figure 4: Performance of combining independently trained LMC1 and LMC2 into LMC3.

In Figure 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 [98] but the latter comes with an order of magnitude less training data. cMNIST-**r** is a variant of the colored-MNIST dataset where the background of 95% of the training samples is colored in **r**ed and 5% in **g**reen. For the cMNIST-**g** dataset these proportions are inverted and 95% of the training samples is colored in **g**reen. The test set contains 50% of samples with green and 50% with red background. We trained LMC1 continually on MNIST, fMNIST+, and cMNIST-r tasks. We trained LMC2 on fMNIST-, cMNIST-g, and SVHN. We then 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 D). Notably, cross-model reusability without fine-tuning is novel to LMC and can be attributed to the local nature of the structural component. Using a global structural component as in [63] would require tuning a separate structural component specifically for LMC3. In case of task-specific routing of proposed for MNTDP [92], additional search would be needed to discover task-specific paths through the consolidated LMC3 model. In both cases the access to the orinal training data distributions would be required and privacy would not be preserved.

4.4 Longer task sequences

Here we study the performance LMC on longer task sequences consisting of 30 – S^{long30}, and 100 – S^{long} tasks. The S^{long} sequence corresponds to the one proposed by Veniat et al. [92] as part of the CTrL benchmark. S^{long30} is a 30-tasks subset of S^{long} (see Appendix B.1 for details).

We first report the average test accuracy ($\mathcal{A}$) and the total number of modules (M) of the models selected through cross-validation. S^{long30}: MNTDP $\mathcal{A}$=64.58, M=64; LMC$_{(\neg A)}$: $\mathcal{A}$=62.44, M=50. S^{long}: MNTDP $\mathcal{A}$=68.92, M=142; LMC$_{(\neg A)}$: $\mathcal{A}$=63.88, M=32. While the gap between the accuracy

of LMC and MNTDP on S^{long30} is only 1.86%-points, in the case of S^{long} this gap grows to 6.58%-points. It is important to highlight that in contrast to LMC, MNTDP's module selection is performed by a task ID aware oracle. We further analyze the trade-off between the number of modules and accuracy in Figure 5, where we plot the number of modules (M) against average test accuracy ($\mathcal{A}$) for models that resulted from training with different hyperparameters. For both streams, we observe that LMC tends to spawn much fewer modules than MNTDP. However, MNTDP shines in the presence of large number of modules and achieves higher overall test accuracy on these streams. Interestingly, as can be clearly observed on the S^{long} stream, LMC reaches higher accuracy with smaller number of modules: e.g. $\sim$64% with 32 modules, while adding modules leads to lower accuracy: e.g. $\sim$61% with 98 modules. This result suggests that local task ID agnostic module selection becomes more challenging for LMC in presence of a large number of modules.

5 Related work

Modularity in neural networks is studied in the context of scalability [9], and more recently as a way to achieve compositionality and systematic generalization [6, 47, 15, 8, 27, 19] as well as for multi-task learning [65, 82]. From the causal point of view, a data generation process could be thought as a composition of independent causal modules [75]. Researchers model these kinds of systems using a set of independent modules, where each module is invariant to changes in the other modules induced by e.g. distribution shifts [84, 76]. This idea is crystallized by Parascandolo et al. [74], who propose a way to learn a set of causal independent mechanisms as mixture-of-experts. Building up on this ideas, others show evidence of compositional OOD generalization [61]. Recently, [66] argue 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 [46, 70, 99], replay [89, 78, 3, 73, 54, 12, 97, 39, 81, 13] or dynamic architectures (and MoEs) [83, 87, 52, 85, 53, 41]. 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 [63, 92]. The main difference with our work is that we use a local composition mechanism instead of a global one. We detail this difference in §2 and also compare to these methods in §4.1.

Continual-meta learning focuses on fast learning and remembering [25, 35, 33, 41], often emphasising the online performance on OOD tasks [14]. As argued by Jerfel et al. [41] 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. Continual-meta learning is often confused with its counterpart *meta-continual learning* [40, 11, 93], in which algorithm are learning to continually learn.

The rapid growth of continual learning has lead researchers to work on empirical studies [20, 58, 56], surveys [32, 42, 55, 66, 67] as well as CL-specific software [72, 22, 59].

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 for each module. It is the locality of the structural component that enables generalization to related but unseen tasks, and that permits 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 direction is removing the need for task boundaries during training and developing more robust architectures for the local structural component (related discussions are in Appendix A.3, and limitations in Appendix G).

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 and Joelle Pineau. Special: Self-supervised pretraining for continual learning. *arXiv preprint arXiv:2106.09065*, 2021.

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

[13] Lucas Caccia, Rahaf Aljundi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. Reducing representation drift in online continual learning. *arXiv preprint arXiv:2104.05025*, 2021.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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`.

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

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

[22] Arthur Douillard and Timothée Lesort. Continuum: Simple management of complex continual learning scenarios. *ArXiv*, abs/2102.06253, 2021.

[23] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[24] 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.

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

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

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

[28] 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.

[29] 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.

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

[31] Sharut Gupta, Praveer Singh, Ken Chang, Mehak Aggarwal, Nishanth Arun, Liangqiong Qu, Katharina Hoebel, Jay Patel, Mishka Gidwani, Ashwin Vaswani, et al. The unreasonable effectiveness of batch-norm statistics in addressing catastrophic forgetting across medical institutions. *arXiv preprint arXiv:2011.08096*, 2020.

[32] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 2020.

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

[34] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A. Rusu, Yee Whye Teh, and Razvan Pascanu. Task agnostic continual learning via meta learning. *ArXiv*, abs/1906.05201, 2019.

[35] 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.

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

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

[38] 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.

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

[40] Khurram Javed and Martha White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems 32*, pages 1820–1830. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8458-meta-learning-representations-for-continual-learning.pdf`.

[41] 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.

[42] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives, 2020.

[43] 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.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

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

[46] 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.

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

[48] 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.

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

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

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

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

[53] 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.

[54] 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.

[55] Timothée Lesort, Massimo Caccia, and Irina Rish. Understanding continual learning settings with data distribution drift analysis, 2021.

[56] Timothée Lesort, Thomas George, and Irina Rish. Continual learning in deep networks: an analysis of the last layer, 2021.

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

[58] Vincenzo Lomonaco, Lorenzo Pellegrini, Pau Rodríguez, Massimo Caccia, Qi She, Yu Chen, Quentin Jodelet, Ruiping Wang, Zheda Mai, David Vázquez, German Ignacio Parisi, Nikhil Churamani, Marc Pickett, Issam H. Laradji, and Davide Maltoni. Cvpr 2020 continual learning in computer vision competition: Approaches, results, current challenges and future directions. *ArXiv*, abs/2009.09929, 2020.

[59] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop, 2021.

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

[61] 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`.

[62] 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.

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

[64] 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.

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

[66] 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, 2020.

[67] Martin Mundt, Steven Lang, Quentin Delfosse, and Kristian Kersting. Cleva-compass: A continual learning evaluation assessment compass to promote research transparency and comparability, 2021.

[68] 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.

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

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

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

[72] Fabrice Normandin, Florian Golemo, Oleksiy Ostapenko, Pau Rodriguez, Matthew D Riemer, Julio Hurtado, Khimya Khetarpal, Dominic Zhao, Ryan Lindeborg, Timothée Lesort, Laurent Charlin, Irina Rish, and Massimo Caccia. Sequoia: A software framework to unify continual learning research, 2021.

[73] 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.

[74] 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.

[75] Judea Pearl. *Causality*. Cambridge university press, 2009.

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

[77] 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.

[78] 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.

[79] 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.

[80] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

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

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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, 2017.

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

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

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

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

[93] Johannes von Oswald, Dominic Zhao, Seijin Kobayashi, Simon Schug, Massimo Caccia, Nicolas Zucchet, and João Sacramento. Learning where to learn: Gradient sparsity in meta and continual learning. *NeurIPS 2021*, 2021.

[94] 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.

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

[96] 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.

[97] 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.

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

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

Acknowledgments and Disclosure of Funding

Laurent Charlin holds a CIFAR AI Chair Program and acknowledges support from Samsung Electronics Co., Ldt., Google, and NSERC. Massimo Caccia was supported through MITACS during his part time employment with Element AI the ServiceNow company. Massimo Caccia was also supported by Amazon, during his part time employment there. We would like to thank Mila and Compute Canada for providing computational resources. We also would like to thank Irina Rish for useful discussions.

A Implementation and Algorithm details

A.1 Batched modularity

Activation weights $w_m^{(l)}$ can be calculated separately for each data point. However, in batched regimes sometimes it can be assumed that module selection for samples in the same batch is likely to be similar.[1] This can be incorporated into LMC by redefining $w_m^{(l)}$ as:

$$w_m^{(l)} = \frac{1}{Z(w)} \text{softmax}(\gamma^{(l)}, \tau)_m \cdot \mu_{w,m}^{(l)}, \tag{9}$$

$$\mu_{w,m}^{(l)} = \mathbb{E}_{x_i \in b}\big(\text{softmax}(\gamma^{(l)}, \tau')_m\big), \tag{10}$$

where $Z(w)$ is the normalization term, b denotes a batch of samples, τ and τ' are the temperature hyperparameters. Lower τ' would results in a stronger bias towards the expected selection $\mu_w^{(l)}$.

A.2 Training with projection phase

The intuition behind the modular training with projection phase is the following: every time a new module addition is triggered (using the mechanism proposed in § 3.1), we start by only adding modules on the deepest layer — i.e. the one closest to the input. Then, in the *projection phase*, we train this module for some time using the signal coming from the structural components of the modules above (possibly combined with the training signal of the downstream task). Projection phase makes sure that the learner first tries to efficiently *reuse* the existing modules (the once above the newly added one) by trying to project it's output into the representation space expected by those modules. After some time the learner is allowed to add new modules again. If the previously added module was not enough to incorporate the distribution shift that caused the previous module addition, new module addition will be triggered in the layers above. We detail this procedure in the Algorithm 1 and 2. Additionally, for modules that recognize current input as outlier in the forward pass, we set their contribution for the current batch to zero (ll.14 in Algorithm 2). This ensures that the newly added modules get enough training signal to learn.

[1] E.g in case of locally stationary data distribution, samples seen together are likely to belong to the same task

Algorithm 1: Modular training with projection

Require: k projection phase length, stream S, z-score threshold z'
Initialize Learner $\mathcal{F}_{\theta,\phi}$
for $t = 0...|S|$ **do**
  $D_t \leftarrow$ get dataset for task t
  for *e = 0 ... total epochs* **do**
    foreach *mini-batch* $b \in D_t$ **do**
      $X, y \leftarrow$ mini-batch of samples
      $\hat{y}, \ell^{str.} \leftarrow$ Forward Model$(\mathcal{F}_{\theta,\phi}, X, z', t)$
      if *new module added in last k epochs* **then**
        `// projection phase`
        $\ell = \ell^{(str.)}$
        if *use functional loss in projection* **then**
          $\ell += \mathcal{L}^{(fnc.)}(\hat{y}, y)$
      else
        $\ell = \mathcal{L}^{(fnc.)}(\hat{y}, y) + \ell^{(str.)}$
      Update parameters θ, ϕ using $\nabla_{\theta,\phi}\ell$
    end
  end
  foreach *module* $m \in \mathcal{F}$ **do**
    Fix structural parameter ϕ of module m
  end
end

Algorithm 2: Forward Model

Require: Learner $\mathcal{F}_{\theta,\phi}$ with L layers, batch X, z-score threshold z', task index t
Output: logits $\hat{y}$, structural loss $\ell^{(st.)}$
$x^{(0)} = X$
$\ell^{(str.)} = 0$
for $l \in L$ **do**
  Let $M^{(l)}$ denote a set of modules at layer l
  Calculate:
  $\gamma^{(l)} \in \mathbb{R}^{|M^{(l)}|\times|X|}$ using Eq. 5,
  $w^{(l)} \in \mathbb{R}^{|M^{(l)}|\times|X|}$ using Eq. 9 or 6,
  $z^{(l)} \in \mathbb{R}^{|M^{(l)}|\times|X|}$ using Eq. 8, average over $|X|$
  $x^{(l)}$ using Eq. 4
  if $t > 0$ **then**
    foreach $m \in M^{(l)}$ **do**
      if $z_m^{(l)} > z'$ **then**
        if *module was added at layer l during task t* **then**
          $w_m^{(l)} = \vec{0}$ `// does not use outlier modules`
        else if ***no*** *module added in last k epochs* **then**
          `// not in the projection phase`
          Fix all modules at layer l
          Add a new free module to layer l
    end
  $\ell^{(str.)} = \ell^{(str.)} - \sum_m sum\{\vec{\gamma}_m^{(l)} \odot \vec{w}_m^{(l)}\}$
end
$\hat{y} = x^{(L)}$
return $\hat{y}, \ell^{(str.)}$

A.3 Structural component

In practice, we applied the log-operation to the structural loss for both choices of the structural component, which resulted in a more stable training procedure.

A.3.1 Invertible architectures

Invertible architectures, such as the one proposed by Dinh et al. [21], can be used to model high-dimensional density after mapping the data in a space with some desirable factorization properties. We use this idea here to directly approximation of the activation likelihood of a module m. More specifically, [37] show that maximizing the likelihood of a module under such invertible transformation is equivalent to minimizing the L_2 norm of the output of structural component $s(o; \phi_m)$, yielding the local structural objective:

$$\mathcal{L}_m^{(str.)}(x) = ||x||^2. \tag{11}$$

To satisfy the invertibility constraint [21] propose to split the input o into blocks of equal size o_1 and o_2 and apply two, not necessarily invertible, transformations s_1 and s_2 as:

$$\begin{aligned} a_1 &= s_1(o_2; \phi_{2,m}) + o_1, \\ a_2 &= s_2(a_1; \phi_{1,m}) + o_2. \end{aligned} \tag{12}$$

The output of structural component a is obtain through the concatenation of a_1 and a_2. Importantly, the input and output of the structural component have the same dimensionality $a_m, o_m \in \mathbb{R}^k$. The inverse can be obtained as:

$$\begin{aligned} o_2 &= a_2 - s_2(a_1; \phi_{2,m}), \\ o_1 &= a_1 - s_1(o_2; \phi_{1,m}). \end{aligned} \tag{13}$$

Intuitively, the invertibility constraint prevents a_1 and a_2 from collapsing to the solution of outputting 0-vectors, which would be useless.

A.3.2 Other possible choices of structural component

The role of structural component in LMC is to detect in-distribution and out-of-distribution samples for each module. It is natural to consider density estimates produced by deep generative models (DGM) for this task. In this work we only considered a simple encoder-decoder based architecture and a simple flow-model. Applying other more complex DGMs such as VAEs [45] or flow-based [10, 48] models might further improve the efficacy of local structural component. Nevertheless, such models also come with their challenges, which include calibration difficulties as well as low data efficiency [95].

A.4 Architecture details

Unless otherwise stated, we initialize the learner with a single module per layer, each learned consist of 4 layers. The used architecture of each module is detailed in Table 2. For the CTrL experiments we used invertible structural component for the task-specific output heads (classifiers) in task ID agnostic LMC(A), while for feature extracting trunk we used an encoder-decoder architecture (i.e. structural component tasked to reconstruct the module's input).

Unless stated otherwise, architectures used for all baselines closely resemble the architecture used by LMC. Thus, by default all baselines contain 4 layers with each layer's architecture and parameter count being equivalent to the architecture and the parameter count of the functional component of an LMC module. In modular baselines (MNTDP, SG-F) each module corresponds to the functional component of the LMC's module. Some of the fixed capacity baselines in Table 1 (e.g. HAT, EWC, Finetune L) where initialized with the layer width scaled up to match the parameter count of the largest possible modular network (e.g. in case of linear growth with one new module per layer per task the largest possible modular network in our framework would have 24 modules in a 6 tasks sequence).

Functional component							
Type	layer $\mathcal{F}$	layer m	#params.	#out ch.	stride	padding	kernel
Conv.	0	0	1792	64	1	2	3
Conv.	1-3	0	36928	64	1	2	3
Batch norm	all	1	128	-	-	-	
ReLu	all	2	-	-	-	-	
Max. Pool	all	3	-	-	-	2	2

Structural component (decoder)							
Type	layer $\mathcal{F}$	layer m	#params.	#out ch.	stride	padding	kernel
ConvTranspose2d	all	0	16448	64	2	2	2
Batch norm	all	1	128	-	-	-	
ReLu	all	2	-	-	-	-	
Conv.	all	3	65600	3	1	1	3
Sigmoid	all	4	-	-	-	-	

Structural component (invertible)					
Type	layer $\mathcal{F}$	layer m	#params.	#input	#output
Input L_2 Norm.	output head	0	-	-	
Linear	output head	1	83232	288	288
Linear	output head	1	83232	288	288

Table 2: Used architectures per module: column 'layer $\mathcal{F}$' refers to the index of the layer in the modular learner $\mathcal{F}$ (i.e. 0 is the closes layer to the input), while 'layer m' gives the layer index within the module. Note, both linear layers in (c) are used in parallel as proposed in [21] (i.e. s_1 and s_2 from Eq. 13).

A.5 Dealing with batch-norms and data normalisation

While batch normalization [38] is a useful device for accelerating the training of neural networks, it comes with challenges when it comes to settings with shifting data distribution such as meta- [7] and continual learning. Specifically, in continual learning when testing on the previously seen tasks after new tasks have been learned, the batch norm will change its statistics resulting in forgetting even if the parameters of the network have not been changed [31]. We highlight several ways to deal with it. One way is to warm-up batch norm before testing on previous tasks by performing several forward passes through the model with unlabeled test data to let batch norms "relearn" the task statistics. This assumes that the test data is available in high quantity at test time (i.e. we cannot warm-up batch-norm with a single test sample). Another way is to fix batch norms completely after a task has been learned. In monolithic architectures, such fixing might limit the plasticity of the network and prevent the learning of new tasks. In modular architectures, however, batch norms of frozen modules

can be kept frozen while new module's batch norms can keep learning resulting in a balance between stability and plasticity. In monolithic architecture which are task-ID aware at test time, a separate batch-norm layer can be used per task.

In modular methods we fixed the batch norms whenever the modules are fixed (e.g. after learning a task in LMC and MNTDP). In HAT we used a separate batch-norm per task, as using a single batch-norm resulted in high forgetting rates. For other non-modular methods (e.g. EWC, ER) we used a single batch-norm layer for all tasks. Additionally, we observed that monolithic methods that share batch-norms across tasks result in high forgetting rate if no data normalisation is performed (and no batch-norm warm-up before testing). In the CTrL experiments we normalised tasks' data for all methods but LMC using statistics calculated on each task separately. In these experiments LMC performed better on not normalised data. In the OOD generalization experiment on cMNIST we normalised data also for LMC. Additionally, we performed batch-norm warm before testing in the OOD experiments. In meta-CL (§ F) the batch norms do not keep the estimates of running statistics ($track_running_stats$ is set to False) and momentum is set to 1.

B Continual Transfer Learning (CTrL)

B.1 Streams used

As in [92], all input samples were reshaped to a 32 x 32 pixels resolution. We normalized for all methods but LMC unless otherwise stated. We did not use any data augmentation techniques. The datasets used for the first 5 streams are described in the Table 3, datasets used for the streams S^{long30} and S^{long} are described in the Table 4 and Table 5.

Stream		T_1	T_2	T_3	T_4	T_5	T_6
S^+	**Datasets**	CIFAR-10[49]	MNIST[51]	DTD[17]	F-MNIST[98]	SVHN[69]	CIFAR-10
	#Train samples	4000	400	400	400	400	400
	#Val. samples	2000	200	200	200	200	200
S^-	**Datasets**	CIFAR-10	MNIST	DTD	F-MNIST	SVHN	CIFAR-10
	#Train samples	400	400	400	400	400	4000
	#Val. samples	200	200	200	200	200	2000
S^{in}	**Datasets**	R-MNIST	CIFAR-10	DTD	F-MNIST	SVHN	R-MNIST
	#Train samples	4000	400	400	400	400	50
	#Val. samples	2000	200	200	200	200	30
S^{out}	**Datasets**	CIFAR-10	MNIST	DTD	F-MNIST	SVHN	CIFAR-10
	#Train samples	4000	400	400	400	400	400
	#Val. samples	2000	200	200	200	200	200
S^{pl}	**Datasets**	MNIST	DTD	F-MNIST	SVHN	CIFAR-10	
	#Train samples	400	400	400	400	4000	
	#Val. samples	200	200	200	200	2000	

Table 3: Details on the datasets and training/validation data amounts for the used streams [92].

B.2 Baselines and training details

We adopted the original soft-gating with fixed modules (**SG-F**) proposed by Mendez and Eaton [63] in two ways: (1) instead or relying on a pool of initially pretrained modules shared across all layers, we initialize a separate set of modules per layer. This is necessary in order to comply with the experimental setup of CTrL which does not allow pretraining. (2) We used the expansion strategy proposed in MNTDP [92] for SG-F, i.e. for each task different layouts with no or one new module per-layer starting at the top layer are trained, the layout with the best validation accuracy is accepted. The original expansion strategy of [63] is similar in spirit, yet relies on a module pool shared between layers and an initial pretraining of modules, which allows training of only two parallel models: with and without adding a single new module to the shared pool.

In **SG-F**$_{(A)}$ we share a single controller network among all tasks in the sequence. Thereby, the main network still uses task IDs to select the task-specific output head. In the controller a single head architecture is used to gate the modules. As modules are added to the learner, each head of the controller used to gate each layer of the main learner is also expanded. This baseline showcases forgetting in the controller if it is shared across tasks.

For **HAT** we used a separate batch-norm layer for each task. Using shared batch-norm resulted in high forgetting rate for this method.

Task	Dataset	Classes	# Train	# Val	# Test
0	CIFAR10	DEER, TRUCK, DOG, CAT, BIRD	25	15	5000
1	MNIST	MNIST 6 - SIX,0 - ZERO,7 - SEVEN,8 - EIGHT,4 - FOUR	5000	2500	4894
2	FASHION-MNIST	COAT, BAG, TROUSER, DRESS, T-SHIRT/TOP	5000	2500	5000
3	SVHN	SVHN 9 - NINE,8 - EIGHT,4 - FOUR,0 - ZERO,6 - SIX	25	15	5000
4	CIFAR100	WORM, POSSUM, AQUARIUM FISH, ORCHID, LIZARD	25	15	500
5	CIFAR10	FROG, AUTOMOBILE, CAT, TRUCK, DOG	5000	2500	5000
6	SVHN	SVHN 3 - THREE,1 - ONE,5 - FIVE,4 - FOUR,7 - SEVEN	25	15	5000
7	MNIST	MNIST 4 - FOUR,5 - FIVE,3 - THREE,2 - TWO,7 - SEVEN	5000	2500	4874
8	FASHION-MNIST	SNEAKER, SANDAL, ANKLE BOOT, COAT, T-SHIRT/TOP	25	15	5000
9	FASHION-MNIST	DRESS, COAT, ANKLE BOOT, BAG, TROUSER	5000	2500	5000
10	SVHN	SVHN 3 - THREE,7 - SEVEN,0 - ZERO,1 - ONE,8 - EIGHT	25	15	5000
11	CIFAR100	OTTER, LEOPARD, BEETLE, RAY, BUTTERFLY	2250	1250	500
12	SVHN	SVHN 6 - SIX,1 - ONE,9 - NINE,2 - TWO,0 - ZERO	25	15	5000
13	FASHION-MNIST	SNEAKER, ANKLE BOOT, T-SHIRT/TOP, SANDAL, DRESS	5000	2500	5000
14	MNIST	MNIST 5 - FIVE,1 - ONE,9 - NINE,7 - SEVEN,8 - EIGHT	5000	2500	4866
15	MNIST	MNIST 5 - FIVE,6 - SIX,7 - SEVEN,9 - NINE,2 - TWO	25	15	4850
16	SVHN	SVHN 4 - FOUR,0 - ZERO,1 - ONE,2 - TWO,7 - SEVEN	25	15	5000
17	FASHION-MNIST	T-SHIRT/TOP, SNEAKER, SHIRT, TROUSER, SANDAL	25	15	5000
18	CIFAR10	CAT, FROG, BIRD, SHIP, DEER	5000	2500	5000
19	SVHN	SVHN 9 - NINE,2 - TWO,8 - EIGHT,4 - FOUR,7 - SEVEN	25	15	5000
20	CIFAR10	SHIP, HORSE, DOG, TRUCK, CAT	25	15	5000
21	FASHION-MNIST	SNEAKER, T-SHIRT/TOP, SHIRT, DRESS, PULLOVER	5000	2500	5000
22	CIFAR10	AIRPLANE, TRUCK, DEER, FROG, BIRD	5000	2500	5000
23	SVHN	SVHN 2 - TWO,6 - SIX,4 - FOUR,1 - ONE,5 - FIVE	5000	2500	5000
24	MNIST	MNIST 8 - EIGHT,3 - THREE,9 - NINE,4 - FOUR,7 - SEVEN	25	15	4956
25	SVHN	SVHN 4 - FOUR,8 - EIGHT,2 - TWO,6 - SIX,7 - SEVEN	25	15	5000
26	SVHN	SVHN 1 - ONE,4 - FOUR,7 - SEVEN,9 - NINE,2 - TWO	25	15	5000
27	CIFAR100	SWEET PEPPER, COCKROACH, MOTORCYCLE, TANK, ELEPHANT	25	15	500
28	SVHN	SVHN 3 - THREE,2 - TWO,4 - FOUR,7 - SEVEN,1 - ONE	5000	2500	5000
29	CIFAR100	CHIMPANZEE, STREETCAR, WOLF, BEAVER, ROSE	25	15	500
30	CIFAR10	HORSE, AIRPLANE, DEER, AUTOMOBILE, TRUCK	25	15	5000
31	SVHN	SVHN 5 - FIVE,8 - EIGHT,7 - SEVEN,4 - FOUR,3 - THREE	5000	2500	5000
32	FASHION-MNIST	COAT, DRESS, SANDAL, PULLOVER, T-SHIRT/TOP	5000	2500	5000
33	CIFAR10	HORSE, SHIP, TRUCK, FROG, CAT	25	15	5000
34	CIFAR10	SHIP, DOG, BIRD, AIRPLANE, CAT	25	15	5000
35	CIFAR10	DEER, AIRPLANE, SHIP, TRUCK, AUTOMOBILE	5000	2500	5000
36	CIFAR100	BOY, BEAVER, WILLOW TREE, SHARK, TANK	25	15	500
37	SVHN	SVHN 3 - THREE,4 - FOUR,9 - NINE,1 - ONE,8 - EIGHT	25	15	5000
38	SVHN	SVHN 9 - NINE,4 - FOUR,5 - FIVE,3 - THREE,1 - ONE	25	15	5000
39	CIFAR10	FROG, AIRPLANE, CAT, DOG, TRUCK	25	15	5000
40	CIFAR10	SHIP, DEER, TRUCK, HORSE, BIRD	25	15	5000
41	FASHION-MNIST	DRESS, SHIRT, TROUSER, COAT, SNEAKER	25	15	5000
42	CIFAR100	STREETCAR, BEAVER, TIGER, BUS, RACCOON	25	15	500
43	FASHION-MNIST	COAT, BAG, DRESS, SNEAKER, SANDAL	25	15	5000
44	MNIST	MNIST 5 - FIVE,9 - NINE,7 - SEVEN,6 - SIX,2 - TWO	5000	2500	4850
45	CIFAR100	HAMSTER, PINE TREE, COCKROACH, BOY, COUCH	25	15	500
46	MNIST	MNIST 0 - ZERO,3 - THREE,2 - TWO,7 - SEVEN,9 - NINE	5000	2500	4980
47	FASHION-MNIST	SANDAL, DRESS, COAT, TROUSER, BAG	25	15	5000
48	SVHN	SVHN 0 - ZERO,8 - EIGHT,5 - FIVE,2 - TWO,1 - ONE	5000	2500	5000
49	CIFAR10	HORSE, FROG, AIRPLANE, DOG, SHIP	5000	2500	5000

Table 4: Details on the datasets and training/validation data amounts used for S^{long} (Part 1) [92].

For all task ID aware methods, the task ID was used either to select the task specific output head (as in HAT, EWC or ER) or the task specific structure as in MNTDP (which includes the output head). Thereby, we treat first and last tasks in S^+, S^-, S^{in} and S^{out} streams as tasks with different IDs. This corresponds to the definition provided in [92], where the task ID is defined to correspond to the sequential order of the task in the sequence.

We used Adam optimizer for all baseline methods but HAT [86]. Using Adam for HAT resulted in more forgetting, which we believe is because HAT masks out only the gradients of some parameters and does not effect Adam's momentum.

Hyper-parameter and model selection was performed using average mean validation accuracy over all tasks in the stream (stream level) with splits detailed in Table 3. When varying the seeds in the provided experiments, we did not very the seed that effects data generation (CTrL Streams) but only the seed that affected the algorithm, model initialization as well as data-loader's batch sampling.

B.3 Metrics

Here we formally define metrics used in the experiments. These metrics are similar to the ones used by Veniat et al. [92]. Δ denotes the prediction accuracy of the predictor $\mathcal{F}$. We use subscripts to indicate the version of the parameters: e.g. $\theta_{1\ldots t}$ indicates the functional parameters of the learner after it was continually trained on t tasks, while θ_T indicates the version of functional parameters after learning only task T in isolation.

Task	Dataset	Classes	# Train	# Val	# Test
50	svhn	svhn 9 - nine,4 - four,6 - six,5 - five,2 - two	25	15	5000
51	svhn	svhn 3 - three,6 - six,8 - eight,9 - nine,1 - one	25	15	5000
52	cifar100	crocodile, lion, butterfly, otter, hamster	2250	1250	500
53	mnist	mnist 9 - nine,8 - eight,6 - six,7 - seven,3 - three	25	15	4932
54	mnist	mnist 7 - seven,3 - three,8 - eight,4 - four,2 - two	25	15	4956
55	svhn	svhn 4 - four,2 - two,6 - six,0 - zero,5 - five	25	15	5000
56	cifar100	sea, chair, snake, spider, snail	25	15	500
57	cifar100	beetle, television, table, porcupine, cup	25	15	500
58	cifar10	cat, horse, frog, truck, automobile	25	15	5000
59	svhn	svhn 8 - eight,6 - six,1 - one,5 - five,3 - three	25	15	5000
60	cifar10	bird, frog, horse, ship, deer	25	15	5000
61	mnist	mnist 1 - one,9 - nine,8 - eight,7 - seven,2 - two	25	15	4974
62	fashion-mnist	Dress, T-shirt/top, Sandal, Trouser, Sneaker	25	15	5000
63	mnist	mnist 6 - six,4 - four,0 - zero,7 - seven,8 - eight	25	15	4894
64	svhn	svhn 4 - four,2 - two,7 - seven,6 - six,3 - three	5000	2500	5000
65	cifar100	pear, skyscraper, shark, plain, dolphin	2250	1250	500
66	cifar10	frog, bird, airplane, ship, horse	25	15	5000
67	cifar10	frog, deer, ship, horse, truck	25	15	5000
68	cifar10	horse, deer, truck, airplane, dog	25	15	5000
69	cifar100	skunk, orchid, cattle, spider, lobster	25	15	500
70	mnist	mnist 3 - three,5 - five,4 - four,9 - nine,1 - one	25	15	4874
71	svhn	svhn 4 - four,3 - three,1 - one,7 - seven,5 - five	25	15	5000
72	fashion-mnist	Coat, Dress, Bag, Sandal, Trouser	25	15	5000
73	fashion-mnist	Sandal, Dress, Ankle boot, Pullover, Shirt	25	15	5000
74	mnist	mnist 3 - three,2 - two,8 - eight,6 - six,4 - four	25	15	4914
75	cifar10	airplane, dog, horse, bird, ship	25	15	5000
76	cifar10	automobile, horse, airplane, cat, truck	25	15	5000
77	fashion-mnist	Sandal, Coat, Shirt, Dress, Ankle boot	25	15	5000
78	fashion-mnist	Trouser, T-shirt/top, Sandal, Sneaker, Dress	25	15	5000
79	cifar100	lion, turtle, cup, shrew, rose	25	15	500
80	mnist	mnist 2 - two,4 - four,5 - five,6 - six,1 - one	25	15	4832
81	cifar100	turtle, mountain, kangaroo, lobster, crab	25	15	500
82	fashion-mnist	Sandal, Sneaker, T-shirt/top, Coat, Pullover	25	15	5000
83	cifar100	plain, skyscraper, butterfly, train, sea	25	15	500
84	mnist	mnist 9 - nine,5 - five,4 - four,8 - eight,2 - two	25	15	4848
85	svhn	svhn 1 - one,7 - seven,0 - zero,5 - five,6 - six	25	15	5000
86	mnist	mnist 2 - two,4 - four,7 - seven,3 - three,8 - eight	25	15	4956
87	cifar10	ship, automobile, frog, dog, horse	25	15	5000
88	cifar100	cloud, spider, tiger, mouse, snake	25	15	500
89	fashion-mnist	Dress, Pullover, T-shirt/top, Bag, Shirt	25	15	5000
90	cifar10	automobile, truck, cat, dog, horse	25	15	5000
91	mnist	mnist 0 - zero,8 - eight,9 - nine,7 - seven,5 - five	25	15	4846
92	mnist	mnist 3 - three,9 - nine,7 - seven,5 - five,8 - eight	25	15	4866
93	fashion-mnist	Bag, Coat, T-shirt/top, Ankle boot, Trouser	25	15	5000
94	cifar100	camel, tractor, orchid, pear, aquarium fish	25	15	500
95	mnist	mnist 2 - two,8 - eight,9 - nine,4 - four,3 - three	25	15	4956
96	mnist	mnist 9 - nine,8 - eight,4 - four,0 - zero,7 - seven	25	15	4936
97	fashion-mnist	Bag, Dress, Shirt, Sandal, Pullover	25	15	5000
98	cifar100	mouse, snail, bed, trout, girl	25	15	500
99	fashion-mnist	Trouser, Pullover, Sandal, T-shirt/top, Ankle boot	25	15	5000

Table 5: Details on the datasets and training/validation data amounts used for S^{long} (Part 2).

Average accuracy on all tasks seen so far.

$$\mathcal{A} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim\mathcal{D}_t}[\Delta\Big(\mathcal{F}(x;\theta_{1\ldots T},\phi_{1\ldots T}),y)\Big)] \tag{14}$$

Forgetting — the average loss of accuracy on a task at the end of training as compared to the first time the task was seen. Positive values indicate positive backward transfer.

$$\mathcal{F} = \frac{1}{T-1}\sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim\mathcal{D}_t}[\Delta\Big(\mathcal{F}(x;\theta_{1\ldots T},\phi_{1\ldots T}),y\Big) - \Delta\Big(\mathcal{F}(x;\theta_{1\ldots t},\phi_{1\ldots t}),y\Big)] \tag{15}$$

Transfer — the difference in performance on the last (T'th) task between the modular learner trained on the entire sequence and an expert $\mathcal{F}'$ trained on the last task in isolation.

$$\mathcal{T} = \mathbb{E}_{(x,y)\sim\mathcal{D}_T}\Delta\Big(\mathcal{F}(x;\theta_{1\ldots T},\phi_{1\ldots T}),y\Big) - \Delta\Big(\mathcal{F}'(x;\theta_T),y\Big) \tag{16}$$

B.4 Transfer results CTrL

	S^-			S^+			S^{in}			S^{out}		
MODEL	ACC t_1	ACC t_1^-	$\mathcal{T}$	ACC t_1	ACC t_1^+	$\mathcal{T}$	ACC t_1	ACC t_1'	$\mathcal{T}$	ACC t_1	ACC t_1''	$\mathcal{T}$
EXPERTS	65.5±0.7	41.8±1.0	0	41.3±2.9	65.6±0.5	0	98.5±0.2	76.9±4.9	0	65.9±0.6	43.5±1.6	0
MNTDP	63.0±3.6	56.9±5.1	15.1	43.2±0.7	65.9±0.8	0.3	98.9±0.1	93.3±1.6	16.4	65.0±1.2	57.7±1.7	14.2
LMC(¬A)	65.2±0.4	60.0±1.1	18.2	42.9±0.9	60.6±1.9	-4.7	98.7±0.1	92.5±7.6	15.6	65.2±0.2	59.8±1.1	16.3
LMC(A)	65.2±0.4	63.0±1.7	21.2	43.1±0.6	62.2±0.7	-3.4	98.7±0.1	88.3±1.6	11.4	65.5±0.6	42.0±21.9	-1.5
S.G+FX	64.9±0.4	49.1±7.3	7.3	43.1±0.4	61.7±1.7	-3.9	98.8±0.1	80.4±6.8	3.5	65.0±0.4	51.5±6.5	8.0

Table 6: Transfer results on the CTrL benchmark. We provide the accuracy of the first and the last task on each of the streams. Additionally, we measure transfer $\mathcal{T}$ as the difference between the last task's accuracy of the model trained on the corresponding stream and an expert model trained on the last task in isolation.

B.5 Module selection: CTrL

In Figure 6 we plot the average module selection of LMC for different streams of the CTrL [92] benchmark after continual training on the corresponding stream. The plots correspond to the runs with the best validation accuracy on the corresponding stream. We observe that for the S^- stream LMC reuses modules trained on the task 1 for the sixth task. On S^+, when tested on task 1 and task 6 same modules are used, yet the two modules at the last layer correspond to the ones added when learning task 6 (this task contains more training data than task 1). On S^{in} the last task reuses only the last layer's module from task 1 and several other modules on the previous layers from previous tasks. Finally, for S^{pl} stream, even though the tasks are unrelated, several modules are reused across the tasks. Importantly, since LMC implements soft module selections strategy several modules can be used together for the same batch of samples (see Eq. 6).

B.6 Ablation of threshold z'

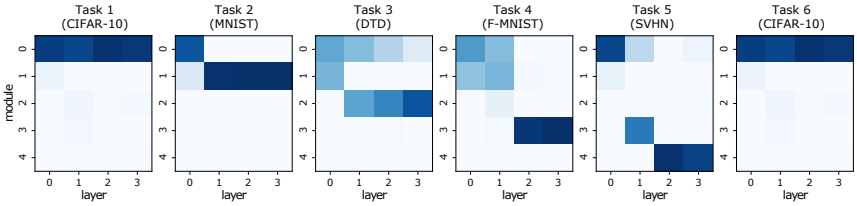
To see how the z threshold influences the final number of modules (M) and the average (test) accuracy over seen tasks ($\mathcal{A}$) we report the $\mathcal{A}$ and M for LMC$_{(\neg A)}$ on 4 streams with fixed hyperparameters while only varying the threshold z'. In Figure 7a we plot the values of the threshold z' on the x-axis against average accuracy $\mathcal{A}$ on the y-axis: higher z' leads to fewer modules being instantiated resulting in lower average accuracy $\mathcal{A}$. Additionally, in Figure 7b we plot the same runs but now with number of modules on the x-axis and the average accuracy $\mathcal{A}$ on the y-axis. We identify the number of modules and accuracy of the MNTDP baseline with dotted lines with the corresponding stream colors in both plots. In Figure 7b we observe that LMC instantiates a comparable number of modules as MNTDP in order to reach similar accuracy, with an exception of the $S^{pl\cdot}$ stream, where LMC tends to add mode modules to reach a similar accuracy.

C Long sequences hyperparameter search visualization

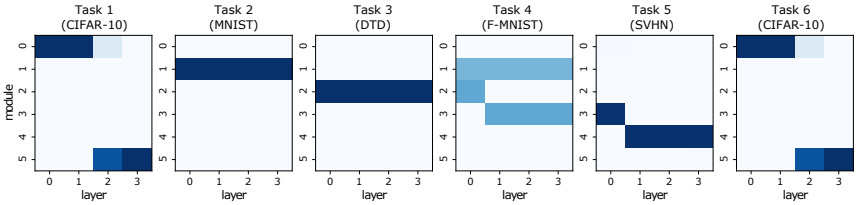
We plot number of modules against the average test accuracy over all seen tasks ($\mathcal{A}$) in Figure 8 for all executed hyperparameter search runs on both S^{long30} and S^{long} sequences.

D Combining independent models

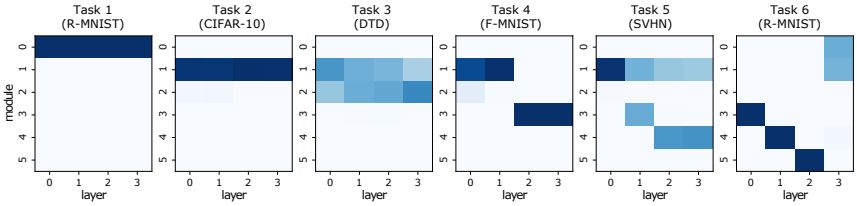
The aim of this experiment is to show the ability of independently trained LMC models to be combined without fine-tuning, without loss in performance and also enabling positive transfer. In this experiment the test set's distributions of the cMNIST tasks are different from the training/validation sets' distributions (see § 4.3). We used oracle model selection choosing hyperparameters on the test set. Note, that model selection is an unsolved challenge in the OOD generalization literature, where selecting model using oracle strategies is sometimes excepted if the baselines methods are also tuned using oracle strategies [30, 77].



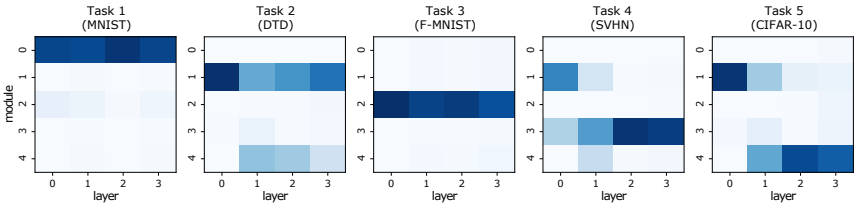
(a) Module Selection S^{-}



(b) Module Selection S^{+}



(c) Module Selection S^{in}



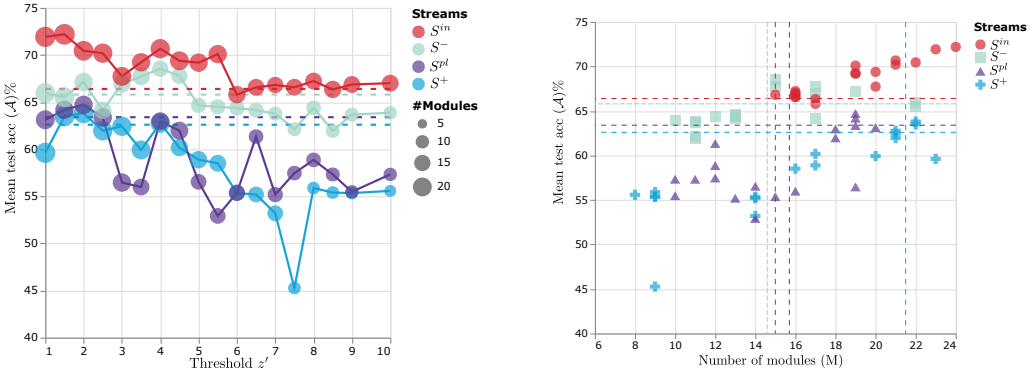
(d) Module Selection S^{pl}

Figure 6: Average module selection for different streams at test time. Horizontal axis corresponds to the layer number, vertical axis corresponds to the module index at a layer; darker color corresponds to higher average activation strength (averaged over batches in the corresponding task).

E Compositional OOD generalization

Some examples of the raw samples used for this experiment are presented in Figure 10. Model selection was performed assuming access to the test sets of the OOD tasks. As mentioned in § D, this is sometimes an excepted strategy is settings in which test distribution is different from the training/validation distribution. Additionally, the input samples were normalized using statistics computed on training set of each tasks, including the OOD tasks (for which no training was performed).

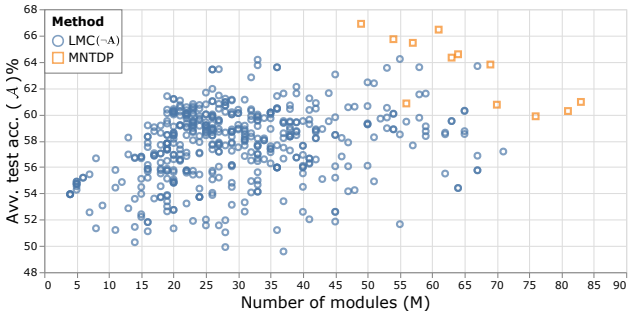
In the main paper we presented a version of LMC with **omitted projection phase**. More precisely this means that the structural loss of the modules above was not propagated into the free modules on the lower layers (closer to the input) and that module addition was allowed during the whole training process.



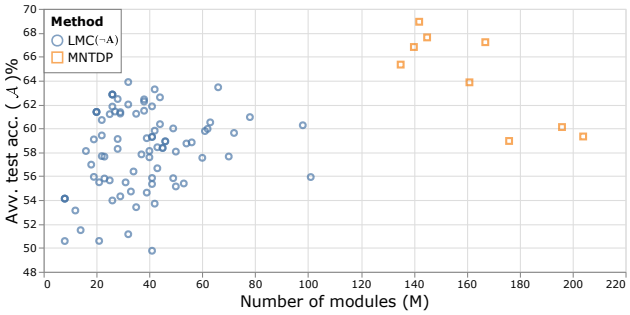
(a) $\mathcal{A}$ and M for different z'.

(b) $\mathcal{A}$ for different M

Figure 7: (a) Mean test accuracy $\mathcal{A}$ and number of modules M for different values of the threshold hyperparameter z' for LMC$_{(\neg A)}$. Dotted lines mark the accuracy of MNTDP[92]. (b) Number of modules (x-axis) against average accuracy $\mathcal{A}$ for runs with different threshold z'. Dotted lines mark $\mathcal{A}$ and M for MNTDP[92] (best seen in color).



(a) 30 tasks sequence S^{long30}.



(b) 100 tasks sequence S^{long}.

Figure 8: Results on S^{long} and S^{long30} sequences for different hyperparameter values for all executed hyperparameter search runs (this is an expanded version of Figure 5).

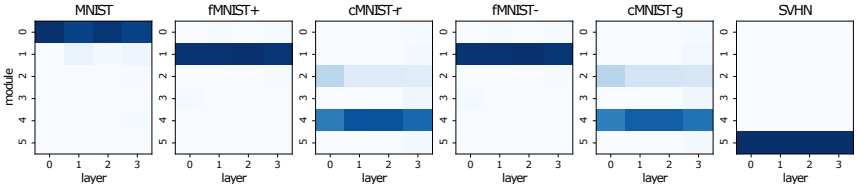


Figure 9: Average module selection per task on LMC3, which was constructed from plugging together independently trained LMC1 (first 3 tasks) and LMC2 (last 3 tasks).

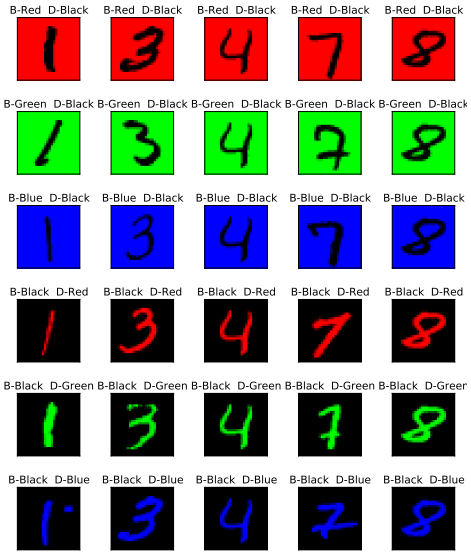


Figure 10: Examples of samples from the used colored-MNIST dataset.

E.1 Module selection

In Figure 11 we plot average module selection per task for the compositional OOD generalization setting.

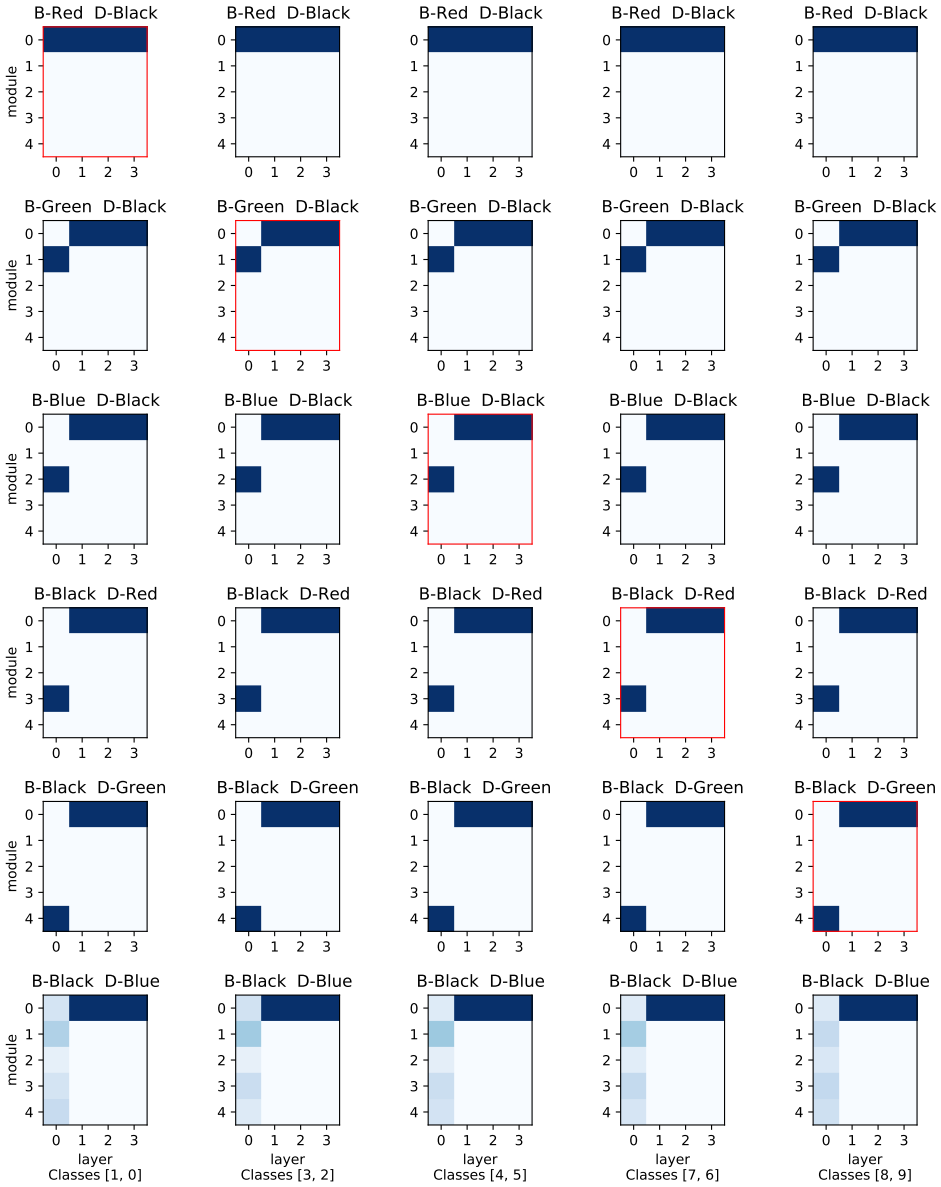


Figure 11: Average module selection per task for compositional OOD generalization setting presented in § 4.2. Each column represents a task (e.g. 0vs.1 classification), each row corresponds to a digit-background color combination. Each square represents a modular learner, x-axis is the layer, y-axis is the modules at each layer. The color intensities correspond to the average activation strength of the module when tested on the corresponding task (the darker the higher). **B** - stands for Background and **D** - stands for digit (e.g. "B-Blue D-Black" stands for blue background and black digit). The diagonal tasks (with red boarder) are seen during the continual training phase, after which the model is evaluated on all 5 combinations of seen digit-background colors for each task. The last row corresponds to the module selection for tasks with digit-background color combination which was not seen during the continual learning phase (black background+blue digit).

F Continual meta-learning

In the continual meta-learning setting, a learner is exposed to tasks sampled from a sequence of environments. The goal of this setting is to construct a learner that quickly (i.e. within a few steps of gradient descent) learns tasks from new environments and relearns (or remembers) tasks from

previously learned environments [41, 14, 34]. Methods applicable in this setting usually rely on gradient-based meta-learning strategies based on MAML [24]. The goal of MAML is to learn a good parameter initialization θ_0 such that the learner can achieve low loss $\mathcal{L}_\tau$ on a randomly sampled task τ after a few steps of gradient descent. The objective of MAML can be formulated as [71]:

$$\theta_0 = arg \min_{\theta} \mathbb{E}_\tau(\mathcal{L}_\tau(U_\tau^k(\theta))), \tag{17}$$

where where U is the operator that performs k inner-loop SGD updates starting with θ using data samples from task τ. MAML has been extensively used in the few-shot learning scenario [94], where similar tasks are sampled from a stationary distribution. However, as pointed out by Jerfel et al. [41], it is unrealistic to assume the existence of a single set of meta-parameters that is close to all tasks in heterogeneous settings with outliers and non-stationary task distributions. In such setting, dissimilar tasks worsen generalization and non-stationarity causes catastrophic forgetting [46]. Instead of learning a single set of meta-parameters for all the tasks, Jerfel et al. [41] propose to learn a set of monolithic expert models, each representing a separate parameter initialization. The meta-learner then leverages the connection between gradient-based meta-learning and hierarchical Bayes [29] for selecting the most likely expert for the given task at hand. The probability of an expert to be selected is calculated based on the inner-loss of each expert as well as a non-parameteric Dirichlet process prior estimated in an online fashion.

Here, we construct two environment sequences. In the first one, the learner is meta-trained on a sequence of Omniglot [50], MNIST [51] and fashion-MNIST [98] 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. The second sequence of environments is constructed from classes of the *mini*Imagenet [79].

We compare to the following baselines. **MAML**: non-modular MAML [24]. **MAML + ER**: MAML with experience replay — we maintain a memory buffer of tasks sampled from previously seen environments, using a reservoir sampling procedure we retrieve a set of previously seen tasks which we then add to the task batches used for meta-training on new environments; **Meta-gating** tries to learn the relevance scores $\gamma^{(l)}$, used for gating, directly in the inner-loop. This is similar to soft-layer ordering used in [63]; **Experts**: an upper bound where separate monolithic model is trained for each environment and selected using the environment ID. We test LMCs with different levels of modular granularity: **LMCe** refers to a model that trains separate experts, but expert selection is performed using an expert-level structural component; **LMCm** is the modular LMC with module selection performed at each layer (as described in § 3). In our experiments we meta-learn the structural components: in the inner-loop we update functional parameters θ_m of all modules (also the fixed ones) whereas the structural parameters ϕ_m are kept constant, in the outer-loop both functional and structural parameters of the free modules are updated. *Importantly, we do not apply the projection phase in the meta-learning experiments.* All experiments are performed in a 5-way 5-shot setting.

For LMCm we used encoder-decoder architectures in the *mini*Imagenet experiments (§ F.3), while for the OMF sequence the invertible structural component was used for the entire network. Note, that task inference is done automatically using the provided meta-test train (query) data, hence, there is no need to use multiple classification heads. For the OMF task sequence each module consists of a single convolutional layer with 64 3x3 filters (padding 1), batch-normalization, ReLU activation function and a max-pooling layer with the kernel-size of 2. This resembles the architecture introduced by Finn et al. [24]. One epoch of meta-training consists of 100 meta-updates, each performed on a batch of 25 tasks (5-way, 5-shot). For the *mini*ImageNet experiment, a single convolutional layer contains 32 filters. One epoch of meta-training performs 100 meta-updates, each on a batch of 4 tasks (5-way 5-shot regime). All learners contain 4 layers, non-modular (MAML, MAML+ER) learners do not expand. In the "Experts" baseline, each expert network corresponds to a 4-layered net with a single module per layer.

Model selection was performed using average meta-test validation accuracy over all environments. We randomly selected 10% of the train datasets for validation purposes for MNIST and fMNIST datasets. For these datasets we did not evaluate the meta-generalization ability of the model, since the classes in meta-train and meta-test splits are the same. For the Omniglot dataset we selected 100 classes for validation purposes. We also flipped the background color of the Omniglot dataset to be

black, which corresponds to the background colors of the MNIST and fMNIST datasets. For the *mini*ImageNet dataset we split the 100 classes in 64-train/16-validation/20-test as in [79].

F.1 Module addition continual meta-learning.

In this section we explain some details about the adaptation of LMC's expansion strategy (§3.1) for continual meta-learning.

The decision about module addition can be made per sample, per task, or per batch of tasks. In the per sample case, a new module is added whenever a sample is regarded as an outlier by all modules at a layer. In the per task case, a new module is added whenever the average z-score for a task is larger than a threshold. In the per batch of tasks case, the average z-score is calculated over a batch of tasks. In our experiments we found that adding new modules on a per batch of tasks base yields the best efficacy in the continual meta-learning setting.

We do not freeze the modules at the environment switch, but create a checkpoint of each non-frozen (free) module. Thus, modules are allowed to learn in the outer loop until the module addition is triggered. Whenever the module addition is triggered at a layer, the existing free module is dropped back to its state from the most recent checkpoint (i.e. its state at last environment switch).

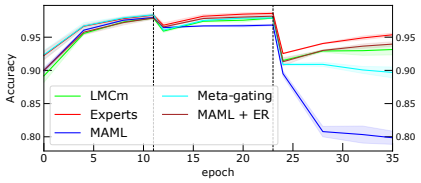
F.2 OMF results

Results on OMF are shown in Figure 12. 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.

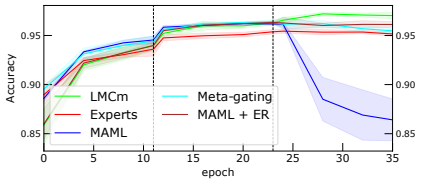
Results with mixtures of tasks are in Figure 12b. We find that LMC attains better performance than the other baselines. The performance improvement is due to LMC assigning modules on a per-sample basis in each forward pass, effectively guiding the inner- and outer-loop gradients of the meta-training procedure to the modules with higher activation on the current input. This contrasts the selection mechanism proposed by [41] for mixture-of-experts, 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.

In Figure 13 we show how the meta-test test accuracies evolve over the course of continual meta-training for each environment of the OMF sequence. Additionally, in Figure 14 we plot the average module selection at meta-test test time on the OMF sequence after the entire sequence has been learned.



(a) Average meta-test accuracy so far



(b) Mixtures of tasks meta-test accuracy

Figure 12: Results on the OMF sequence. Vertical dashed lines mark the environment switches.

F.3 Additional *mini*Imagenet results.

In this section we present additional results on the evolving *mini*Imagenet sequence where, similarly to [41], each environment is obtained through application of filters 'blur', 'pencil' and 'night'. Figure 16 plots per environment accuracy of each method that results in the average accuracy over the environments seen so far depicted in Figure 15 (a).

Additionally, we design a mixture-task baseline as follows: we select randomly 5 classes from the *mini*ImageNet dataset, sample datapoints for these 5 classes from each of the environments seen so

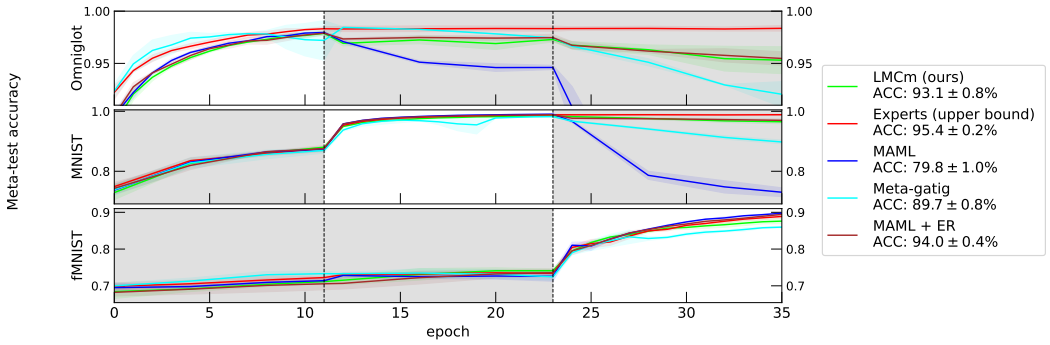


Figure 13: Meta-test test accuracy on the OMF task sequence. Each row shows accuracy on a specific environment, the training time on each environment is highlighted with white background. We also report the average accuracy over all environments at the end of training in the legend (ACC). For this experiment the modular learner (LMCm) was trained using inevitable network as structural component (5-way 5-shot).

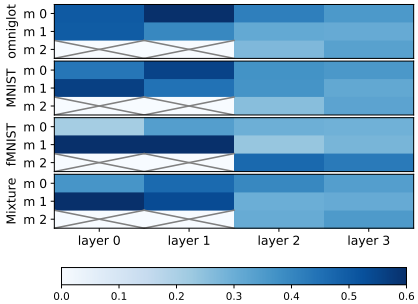
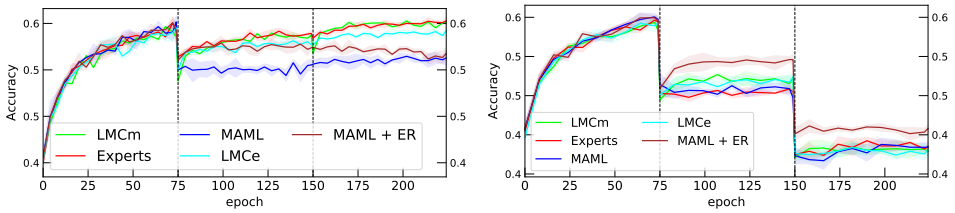


Figure 14: Average module selection of LMCm for a selected run on the OMF task sequence (after continual meta-CL training has been complete on the entire sequence).



(a) Average meta-test accuracy so far

(b) Average meta-test accuracy on the mixture tasks

Figure 15: Meta-test test results on the sequence of *mini*Imagenet environments. Vertical lines mark environment switches.

far (i.e. all three at the end of the sequence) to build tasks. Hence, a single mixture task for each class will contain samples from each environment seen so far. The learner is then meta-tested on these tasks. This is different from the mixture task designed for the OMF sequence, where classes from different environments were mixed. We present the results on the mixture task in Figure 15(b). We observe that MAML+ER baselines outperforms other learners in this evaluation setting. We hypothesize that poor performance of LMC in this setting is due to the batched modularity procedure (§A.1), where biasing module activation towards majority activation withing the batch of may harm performance, since each batch contains samples from all environments seen so far.

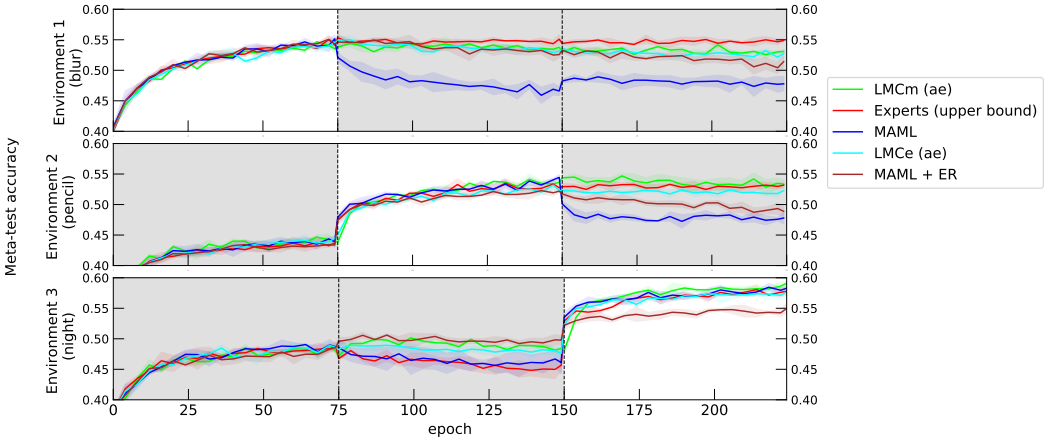


Figure 16: Meta-test test accuracy on the evolving *mini*ImageNet task sequence. Each row shows accuracy on a specific environment, the training time on each environment is highlighted with white background. For this experiment the modular learners (LMCm and LMCe) were trained using decoder as structural component tasked to reconstruct modules' inputs (5-way 5-shot).

G Limitations

We identify three main limitations of LMC. The first one is that its computational footprint is not constant with respect to the number of tasks. This is caused by the fact that each forward path through the modular learner requires a forward pass through each module in order to obtain the local relevance scores. Potentially, this can be overcome by assuming local stationarity and using the same modules for several consecutive batches of samples. On the other hand, this might not be a big problem at all: we think of each module as an independent entity that can be executed on it's own dedicated device, in which case at each layer each module's computations can be executed in parallel. The recent trend towards shifting from monolithic to modular architectures can accelerate the development of such hardware devices.

Another limitation of LMC is the lack of direct communication between modules at the same layer, which was shown to be important in some situations such as when different physical processes interact [27, 28]. While explicitly modeling such communication is an interesting direction for the future work, the consolidation of modules through weighed sum in Eq. 4 can be though of as a form of implicit communication, i.e. a form of a shared workspace similar to the one presented in [28]. It is however questionable whether cross-module communication is of any benefit in a standard supervised-learning settings.

Additionally, as discussed in § 4.4, module selection becomes challenging for LMC in presence of large number of candidate modules resulting in lower accuracy on long task sequences as compared to oracle based sleection strategy implemented in MNTDP [92].

Finally, in its working LMC relies on the local OOD detector as well as a generative model as it's structural component. Several recent works have shown that deep generative models often mistakenly assign high likelihood values to outlier points [68, 36, 95]. Hence, LMC's success depends on overcoming these issues through innovation in the fields of OOD detection and generative modeling.

H Broader societal impact

This work aimed at leveraging modularity and compositionality for continual learning (CL). The goal of CL is to design systems capable to retain knowledge and transfer knowledge across tasks. Such systems can **positively impact** society in the following ways: (i) models able to retain knowledge withing neural connections do not require storing raw samples in a replay buffer, yielding systems that are more compliant with data privacy standards. (ii) Positive transfer of knowledge across tasks can result in more resource efficient training. (iii) Building modular systems can further improve resource

efficiency: e.g. as shown in the experiment in § 4.3 several modular systems can be combined in a third system without any retraining.

We do not identify any potential **negative societal impacts** of this work in particular beyond the potential negative societal impacts of artificial learning systems in general, which include the risk of decision bias, loss of certain jobs due to automation, risk of increased vulnerability to hacker attacks to name a few.