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 code base 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 [61]. CL is often modeled as a trade-off between knowledge retention (stability) and knowledge expansion (plasticity) [26, 63]. 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 [45, 56]. Conversely, learning a separate model per task eliminates forgetting but minimizes transfer and data efficiency [2, 40].

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, 46, 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 [46, 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 [90, 89, 94].

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

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

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 [91, 62]. 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 [91] or as a model- and task-specific controller network that generates modular layouts [62]. Unfortunately, in many realistic CL scenarios task identities are unavailable at test time [23, 34, 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 [91] (§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 [91] 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 [91, 45]. The goal is to learn an optimal θ^* that minimizes the loss $\mathcal{L}$ for all observed tasks:

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

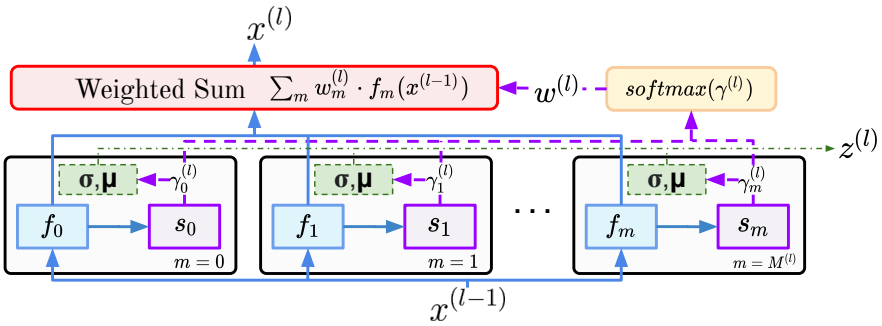


Figure 1: **Modular Layer Scheme**. Each black rectangle is a module. Inside each module, the functional component f_m receives the input $x^{(l-1)}$ and feeds its output to the structural component s. The output of the structural component is used to calculate the importance score $\gamma_m^{(l)}$ using 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 [45, 86, 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, 82]. 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. [91] 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 [62, 81, 46, 64]. 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 [62] 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.

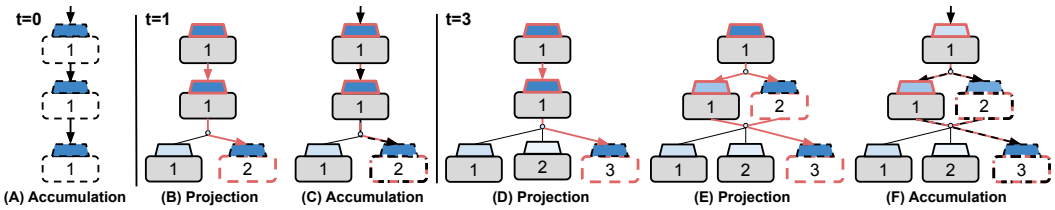


Figure 2: **Two-phase training**. Each module contains a functional (rectangle) and structural (trapezoid) component. Their color intensity denotes the strength of their activation. Components with dashed contours are trained, solid contours represent fixed components, arrows show the gradient flow: 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 [81], or alternatively selecting top-k modules [87]. 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 [46] or reinforcement-learning based methods [81, 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 [91], 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 [79]. Here we use the invertible architecture proposed by Dinh et al. [21]. As shown by Hocquet et al. [36], 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[86]	54.5±1.3	-0.6±0.6	24*	51.1±2.3	-0.8±1.2	24*	48.0±1.4	-2.3±0.9	24*	54.4±2.4	-0.6±0.7	24*	56.0±0.9	-2.0±1.1	20*
EWC[45]	61.0±0.8	-5.9±1.2	24*	53.2±1.0	-5.8±1.2	24*	60.2±1.0	-4.6±1.1	24*	60.8±1.0	-6.0±1.0	24*	54.3±1.0	-2.7±1.4	20*
O-EWC[84]	59.1±3.3	-8.4±4.1	24*	52.6±1.9	-5.8±1.2	24*	59.7±1.0	-5.0±1.0	24*	61.3±2.4	-6.6±2.3	24*	50.8±1.9	-7.6±2.0	20*
ER[80, 16]	62.5±0.6	-1.4±0.7	4*	61.8±0.8	3.9±1.0	4*	57.7±2.3	3.2±1.2	4*	62.4±0.7	-0.9±0.5	4*	62.4±0.4	3.0±0.3	4*
EXPERTS	62.6±0.7	0.0	24	**62.8±0.9**	0.0	24	65.2±1.0	0.0	24	63.0±0.5	0.0	24	**64.2±0.6**	0.0	20
MNTDP[91]	65.8±0.6	0.0	14.6	62.6±1.0	0.0	21.5	66.4±0.9	0.0	15.0	66.2±0.8	0.0	13.7	63.4±0.5	0.0	15.7
SG-F[62]	63.6±1.1	0.0	14.7	60.5±0.5	0.0	18.5	63.9±1.5	0.0	19.2	63.6±1.8	0.0	14.3	62.8±1.4	0.0	16.7
LMC(¬ A)	**66.6±1.5**	-0.0±0.1	15.3	60.1±2.7	-1.4±2.4	21.3	**69.5±1.0**	0.0±0.1	20.0	**66.7±2.2**	-0.1±0.1	15.5	61.6±4.8	-3.5±3.1	18.2
MNTDP(A)	44.5±3.0	-5.5±1.9	15.3	43.1±4.2	-9.2±5.4	20.7	32.7±13.6	-15.2±1.3	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	**60.5±4.9**	-4.4±4.7	18.2
LMC(A,H)	64.9±1.5	-0.2±0.2	16.2	55.8±2.5	-0.3±1.2	15.3	67.6±2.7	-0.8±1.0	21.5	54.2±3.6	-2.9±2.0	15.9	53.8±5.7	3.1±5.5	10.8
SG-F(A)	29.5±3.5	-35.3±4.0	14.3	20.4±4.4	-39.3±6.7	16	24.4±5.6	-38.7±4.0	18.7	30.5±4.5	-34.0±5.5	12.2	19.4±1.0	-41.8±1.6	15.5
ER(A,S)[80, 16]	60.4±0.7	1.0±1.3	4*	61.9±0.6	5.3±0.8	4*	57.7±1.0	5.0±3.5	4*	50.7±0.8	-0.7±1.8	4*	60.0±0.7	1.2±1.5	4*
FINETUNE	47.2±3.7	-14.6±3.8	4*	30.5±2.8	-28.4±2.9	4*	42.4±6.0	-22.5±6.5	4*	47.1±2.5	-14.0±2.7	4*	29.6±2.0	- 28.6±2.1	4*
FINETUNE L	52.1±1.4	-15.7±1.7	24*	38.3±2.7	-24.0±2.8	24*	51.2±1.6	-13.3±1.8	24*	51.3±1.9	-17.3±2.1	24*	38.6±2.4	-24.1±2.2	20*

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

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 using the Continual Transfer Learning Benchmark (CTrL) [91]. 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 [91]. 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 (larger 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** [91]: 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** [62]: 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 net shared across tasks. **HAT**[86]: 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** [45]: trains a single model for all tasks while applying parameter-regularization to minimize forgetting. **O-EWC**:[84] 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 5-task sequence). **ER**$_{(A,S)}$: task ID agnostic version of ER that uses a single output head to classify all classes from all tasks: 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.

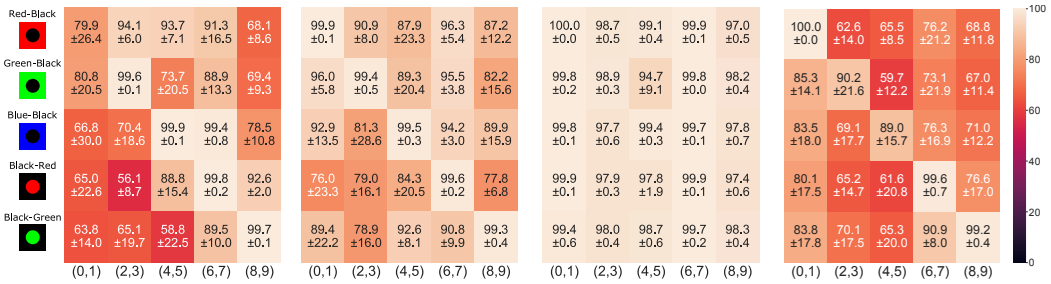
We use several versions of LMC. **LMC**$_{(\neg A)}$ a version of LMC that uses the task ID for output head selection (not module selection as MNTDP). **LMC**$_{(A)}$: the default version of LMC. It equips output heads with structural components and is therefore task ID agnostic at test time. **LMC**$_{(A,H)}$: a version of task ID agnostic LMC that performs hard module selection, i.e., taking the module with the highest relevance score per layer. Architectural and training details are in Appendix B.2. All methods use the same architecture (described in Appendix A.4) together with the Adam [43] optimizer. HAT is the only method that uses SGD.

Similar to Veniat et al. [91], 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 [59]; **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 [62]. HAT showed low forgetting, however was not able to learn the tasks well, which we attribute to the use of the SGD optimizer as opposed to Adam used by other methods. A version of LMC performs the best on the S^+, S^{in} and S^{out} streams.

Notably, LMC$_{(A)}$, which does not rely on task IDs, outperformed all other task ID agnostic methods such as MNTDP$_{(A)}$ and ER$_{(A,S)}$, and performed on par with task ID aware methods on all streams but S^{out}. Low performance is expected for methods relying on automated task inference on the S^{out} stream due to output distribution shift: i.e., at test time we notice that LMC correctly assigns samples from the last task t_1'' to the first task's t_1^+ output head. However, the resulting classification accuracy is low because the labels were permuted for the last task.

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 t_1^+, leading to lower accuracy. In addition, LMC exhibits high variance on the S^+ and S^{pl} streams leading to it being outperformed by



(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).

Experts and the MNTDP. This may be caused by the larger amount of trainable parameters compared to other models. Finally, low performance of LMC$_{(H)}$ emphasizes the importance of soft modular attention for LMC. Additional results, including on a transfer metric [91], are in Appendix B.4.

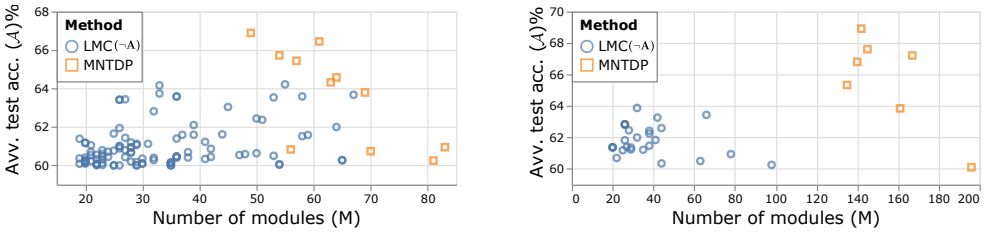
4.2 Compositional OOD generalization

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 [42] 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 [45], MNTDP [91], 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 optumal 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.

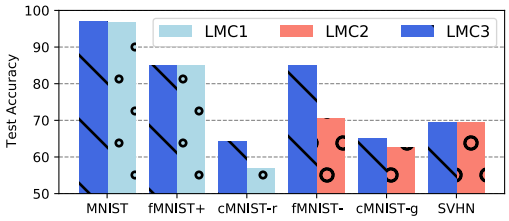


Figure 4: Performance of combining independently trained LMC1 and LMC2 into LMC3.

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 [96] 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. LMC2 was trained 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 [62] would require tuning a separate structural component specifically for LMC3. In case of task-specific routing of proposed for MNTDP [91], 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. [91] 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 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, 46, 15, 8, 27, 19] as well as for multi-task learning [64, 81]. From the causal point of view, a data generation process could be thought as a composition of independent causal modules [74]. 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 [83, 75]. This idea is crystallized by Parascandolo et al. [73], 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 [60]. Recently, [65] 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 [45, 69, 97], replay [88, 77, 3, 72, 53, 12, 95, 38, 80, 13] or dynamic architectures (and MoEs) [82, 86, 51, 84, 52, 40]. 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 [62, 91]. 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, 34, 32, 40], often emphasising the online performance on OOD tasks [14]. As argued by Jerfel et al. [40] 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* [39, 11], in which algorithm are learning to continually learn.

The rapid growth of continual learning has lead researchers to work on empirical studies [20, 57, 55], surveys [31, 41, 54, 65, 66] as well as CL-specific software [71, 22, 58].

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

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

[33] 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.

[34] 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.

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

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

[37] 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.

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

[39] 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`.

[40] 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.

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

[42] 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.

[43] 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.

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

[45] 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.

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

[47] 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.

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

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

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

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

[52] 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.

[53] 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.

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

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

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

[57] 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.

[58] 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.

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

[60] 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.

[61] 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.

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

[63] 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.

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

[65] 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.

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

[67] 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.

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

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

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

[71] 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.

[72] 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.

[73] 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.

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

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

[76] 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.

[77] 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.

[78] 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.

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

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

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

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

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

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

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

[92] 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.

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

[94] 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.

[95] 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.

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

[97] 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.