Harnessing Neural Unit Dynamics for Effective and Scalable Class-Incremental Learning

Depeng Li [1] **Tianqi Wang** [1] **Junwei Chen** [1] **Wei Dai** [2] **Zhigang Zeng** [1]

Abstract

Class-incremental learning (CIL) aims to train a model to learn new classes from non-stationary data streams without forgetting old ones. In this paper, we propose a new kind of connectionist model by tailoring neural unit dynamics that adapt the behavior of neural networks for CIL. In each training session, it introduces a supervisory mechanism to guide network expansion whose growth size is compactly commensurate with the intrinsic complexity of a newly arriving task. This constructs a near-minimal network while allowing the model to expand its capacity when cannot sufficiently hold new classes. At inference time, it automatically reactivates the required neural units to retrieve knowledge and leaves the remaining inactivated to prevent interference. We name our model AutoActivator, which is effective and scalable. To gain insights into the neural unit dynamics, we theoretically analyze the model's convergence property via a universal approximation theorem on learning sequential mappings, which is under-explored in the CIL community. Experiments show that our method achieves strong CIL performance in rehearsal-free and minimal-expansion settings with different backbones.

1. Introduction

Contrary to typical machine learning methods that work on independent and identically distributed data, class-incremental learning (CIL) tackles the problem of training a single model on non-stationary data distributions. In this scenario, tasks typically consist of subsets of disjoint classes that are presented sequentially, without providing task identities at inference time (Wang et al., 2023). However, with data of the current task accessible but none (at least the bulk) of the past, CIL is challenged by a sharp performance decline on the previously learned tasks, known as catastrophic forgetting problem (McCloskey & Cohen, 1989).

Recently, CIL of neural networks has seen explosive growth in striving for less forgetting (Bonicelli et al., 2022; Tong et al., 2023; Qiao et al., 2024; Li et al., 2024a). Prior works fall into three main categories (Masana et al., 2023). *Rehearsal-based approaches* maintain a small portion of past samples and mix them with that of a new task at either input layer (pixel level) (Lopez-Paz & Ranzato, 2017; Bang et al., 2021) or hidden layer (internal representations) (Van de Ven et al., 2020; Hayes et al., 2020). However, this line of work suffers from substantial performance degradation with a smaller buffer that carries inadequate task-specific knowledge and becomes infeasible when a rehearsal buffer is not allowed due to memory constraints or privacy issues. *Regularization-based approaches* aim to minimize the impact of learning new tasks on the weights (Kirkpatrick et al., 2017; Wołczyk et al., 2022) or feature representations (Schwarz et al., 2018; Li et al., 2024b) that are important for previous tasks. Although avoiding data storage, the involved penalty terms make a fixed-size model rather inflexible to find the optimal solutions as it retains the memory of previous classes entirely in the parameter space. *Architecture-based approaches* dynamically adapt network components by expansion (Verma et al., 2021; Yang et al., 2023) or mask (Serrà et al., 2018; Ke et al., 2021) operation to absorb knowledge for novel classes. Nevertheless, network expansion usually renders the model size grow quickly as each session proceeds, which should be counted into the memory budget for a fair comparison (Zhou et al., 2023).

Yet, in the brain—which clearly has implemented an efficient and scalable function for incremental learning—the *reactivation of neuronal activity patterns* that represent previous experiences is believed to be important for stabilizing new memories (Rasch & Born, 2007; Joseph et al., 2010). Motivated by the principle of learning and memory in cognitive neuroscience, this paper proposes a new kind of connectionist model that automatically reactivates the involved

[1]School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China [2]School of Information and Control Engineering, China University of Mining and Technology, Xuzhou, China. Correspondence to: Zhigang Zeng <zgzeng@hust.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

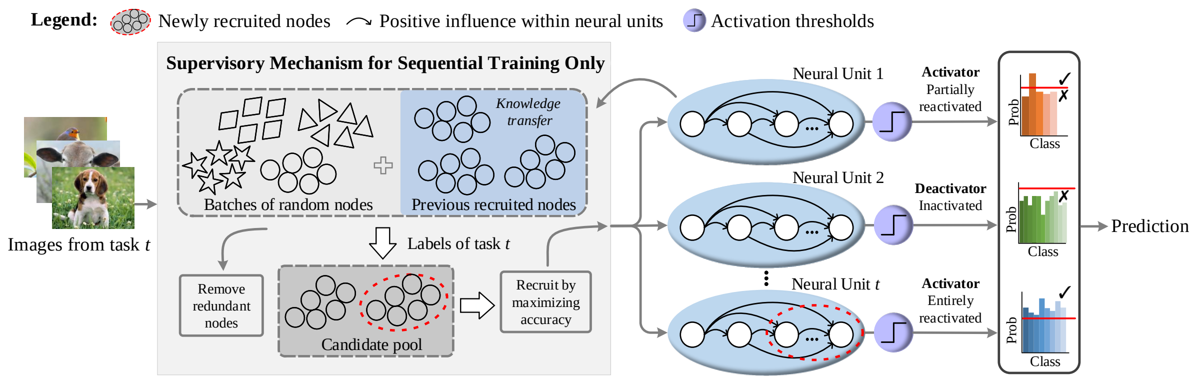

Figure 1. Overview of AutoActivator. It first generates several batches of random nodes, denoted by different shapes; Then, together with existing ones for knowledge transfer, it parsimoniously recruits new nodes meeting the supervisory mechanism (e.g., in red circles) to a scalable neural unit, where those joined ones are positively influenced by each other as marked by black arrows. In AutoActivator, the former layers are built under the guidance of supervisory mechanism (Section 4.1) while the final classifier layer is step-wise updated by close-formed solutions (Section 4.2). The activation thresholds render a neural unit partially/entirely active or inactivated for prediction.

collections of nodes (dubbed neural units)[1] as activators to retrieve knowledge while leaving the remainder as deactivators to avoid crosstalk, hence the name AutoActivator. This is achieved by harnessing neural unit dynamics that adapt the behavior of neural networks for CIL. As shown in Figure 1, AutoActivator pioneers a novel CIL paradigm that runs through the training and test phase:

As opposed to the over-parameterized or expanding-and-pruning implementations, one can start with modeling the neural unit from scratch and dynamically grow the network as actually needed by a given task. Specifically, in each training session, we first randomly allocate several batches of random nodes together with the counterparts from earlier sessions and introduce a supervisory mechanism to remove the redundant nodes for the current session. Those meeting the supervisory mechanism are temporarily on standby in the candidate pool. Then, only one batch that contributes to causing a maximum reduction in training errors is added to the corresponding neural unit, i.e., the recruited nodes are positively influenced by the existing ones such that each plays an irreplaceable role. This way constructs a minimalist network for sequential tasks and thus suffices to train more incoming tasks. Meanwhile, we parameterize each neural unit with an activation threshold measured by the predicted probability during training, which guards the decision boundary of learned classes against distortion. At inference time, given batches of test instances from a certain task or class trained, the reactivation of neural units is performed partially or entirely without knowing the task identities.

Our main contributions are threefold:

- We design neural unit dynamics that govern the behavior of neural networks, including the rules of node generation/connection, activation threshold, and update, as well as interactions responding to non-stationary data streams. The model's convergence property on learning sequential mappings is theoretically guaranteed.
- AutoActivator is an efficient and scalable CIL method, characterized by parsimoniously constructing a CIL model whose complexity is commensurate with the intrinsic complexity of each learning task. The method is inherently immune to catastrophic forgetting as neural units reactivated partially or entirely do not infringe upon others, and has strong task-order robustness.
- Experiments on multiple benchmark datasets consistently demonstrate that our method provides competitive CIL performance, with absolute superiority of rehearsal-free and minimal-expansion desiderata.

[1]Herein the node means a neuron and incoming weights and bias associated with it and neural unit is collection of such nodes.

2. Related Work

Class-Incremental Learning. We discuss a selection of representative CIL approaches and how they relate to our work. Rehearsal-based approaches explicitly preserve data from previously learned tasks to retrain with the current task. Aided with rehearsal buffers, IL2M (Belouadah & Popescu, 2019) rectifies the network predictions, RM (Bang et al., 2021) focuses on the classification uncertainty to select hard samples, and i-CTRL (Tong et al., 2023) is founded on structured representations for rehearsal. Our method does not buffer past data for the whole CIL process and thus eliminates shortcomings such as scalability and privacy issues. Regularization-based approaches involve penalty terms to vary the plasticity of parameters. EWC (Kirkpatrick et al., 2017) is the pioneer of this branch, followed by SI (Zenke et al., 2017), and MAS (Aljundi et al., 2018). Their network parameter is associated with the weight importance computed by different strategies. Nevertheless, the challenge is to correctly assign credit to the network weights when the number of tasks is large. Our method keeps the neural unit weights intact during sequential training, i.e., the newly recruited nodes without infringing upon others. Architecture-based approaches isolate existing model parameters or attach additional parameters as each session proceeds. Methods such as DER (Yan et al., 2021), FOSTER (Wang et al., 2022a), and DNE (Hu et al., 2023) acquire sufficient learning capacity by adding a sub-network per task. However, the increased capacity for future tasks must be meticulously balanced with the number of parameters added, particularly considering that the number of tasks the model needs to learn is often unknown in advance. Our method differs in that (i) the expansion quota is commensurate with the intrinsic complexity of each task; (ii) it's more effective to start with a small/compact branch instead of scaling arbitrarily and pruning; and (iii) empirically, our final memory budget is comparable or even superior to the non-growing networks in regularization-based methods.

Neural Network Dynamics. The learning dynamics of a connectionist model, such as an artificial neural network (ANN), refers to how the network's internal state evolves over time in response to inputs (Vahedian et al., 2021; Márton et al., 2022). This can involve different levels, including the connections of the neurons, the activation patterns, the flow of information through the network's layers, and the overall convergence and learning behavior (Vyas et al., 2020). For example, training a feed-forward neural network with random hidden nodes has been explored in the single-task learning (Pao & Takefuji, 1992; Li & Wang, 2017). The general idea is to randomly generate hidden nodes (weights and biases), and only output weights need to be tuned in either a deterministic or nondeterministic manner. Such a randomized learning dynamic has demonstrated great potential in developing fast learner models and easy-implementation learning algorithms, such as convolutional/graph randomized networks (Zhang & Suganthan, 2016; Huang et al., 2023). By bridging this intriguing learning dynamic to the architectural update paradigm, our work attempts to build an efficient and scalable network for CIL.

3. Class-Incremental Learning Setup

Notations. Denote $\Gamma = \{g_1, g_2, \dots\}$ as a set of bounded nonconstant piecewise continuous functions, span(Γ) as a function space spanned by Γ, and $L_2(D_t)$ as the space of all Lebesgue measurable functions $f = [f_1, f_2, \dots, f_{C_t}] : \mathbb{R}^{M_t} \to \mathbb{R}^{C_t}$ defined on D_t. Hence, L_2 norm is defined as

$$\|f\| = \left(\sum_{c=1}^{C_t} \int_{D_t} |f_c(x)|^2 dx \right)^{\frac{1}{2}} < \infty \tag{1}$$

The inner product of $\vartheta = [\vartheta_1, \vartheta_2, \dots, \vartheta_{C_t}] : \mathbb{R}^{M_t} \to \mathbb{R}^{C_t}$ and f is further formulated as

$$\langle f, \vartheta \rangle = \sum_{c=1}^{C_t} \langle f_c, \vartheta_c \rangle = \sum_{c=1}^{C_t} \int_{D_t} f_c(x) \vartheta_c(x) dx \tag{2}$$

We now define CIL formally. A model sequentially learns from the supervised learning datasets $D_t = \{(X_t, Y_t) | X_t \in \mathbb{R}^{N_t \times M_t}, Y_t \in \mathbb{R}^{N_t \times C_t}\}$ of task t $(t = 1, 2, \dots, T)$, where X_t is the data, Y_t is the label, N_t is the number of samples, M_t and C_t are the dimensions, respectively. The model $\mathcal{M}(X_{t-1}; \theta_{t-1})$ $(t \geq 2)$ trained on previous task(s) is parameterized by its connection weights θ_{t-1}. The objective is to train an updated model $\mathcal{M}(X_t; \theta_t)$ that accommodates the newly emerging C_t classes, during which the data of previous tasks is inaccessible. When fed test instances from any of tasks 1 to T, the model $\mathcal{M}(X_T; \theta_T)$ can make predictions without task descriptors/identifiers.

Fair Comparisons. Since different CIL methods have very different requirements in data, networks, and computation, it is intractable to compare all under the same experimental conditions. Following the suggestion in (Zhou et al., 2023), we holistically evaluate different methods by considering both accuracy and memory cost for a fair comparison. We align the memory cost of model size and exemplar buffer (if any) by switching them to a 32-bit floating number which we refer to as *memory budget*. Also, with the increasing prominence of foundation models, pre-trained models equipped with informative representations have become available for various downstream requirements (McDonnell et al., 2023; Mehta et al., 2023). Following the settings in prior work (Cha et al., 2021; Rios et al., 2022; Bonicelli et al., 2022; Tang et al., 2023), we optionally inject AutoActivator into some advanced pre-trained backbones such as ResNet (used in PCL (Hu et al., 2021), OWM (Zeng et al., 2019), etc.) and ViT (used in DualPrompt (Wang et al., 2022c), CODA-Prompt (Smith et al., 2023), etc.). Such a setting accommodates real-world scenarios where pre-training is usually involved as a base session (Bonicelli et al., 2022). We conduct experiments across multiple datasets with or without starting with the same pre-training.

4. Methodology

Our AutoActivator is a new kind of connectionist model (see Figure 1), with tailored rules of node generation/connection, activation threshold, and update, as well as interactions responding to sequential tasks. To obtain a good grasp of the CIL model, we first provide the theoretical guide to node expansion in Section 4.1. We then perform the reactivation of involved neural units for learning-without-forgetting decision-making in Section 4.2.

4.1. Modeling Neural Units via Supervisory Mechanism

Instead of empirically over-parameterized or expanding-and-pruning implementations, we seek a brand-new solution with theoretical support during sequential training. With this consideration, we start with modeling a neural unit from scratch and then grow additional nodes as the given problem requires. In this way, the expansion quota is compactly commensurate with the intrinsic complexity of each task, and thus constructs near-minimal neural network architectures for the CIL process. However, two main problems are: (1) How to generate and connect new nodes to a scalable neural unit? (2) How to update and then reactivate the neural units recruited without recourse to task identities?

To answer the first question, we draw inspiration from recent advances in network randomization (Huang et al., 2023; Ramanujan et al., 2020; Wang & Li, 2017), a randomized learning technique for developing fast learner models and easy-implementation learning algorithms. A common and basic idea behind this technique is to randomly generate hidden nodes (weights and biases), and only output weights need to be tuned (Pao & Takefuji, 1992; Zhang & Suganthan, 2016; Li & Wang, 2017; Zhang et al., 2022). In this way, one can add new nodes with random weights to different network layers progressively. Specifically, given a target function $f:\mathbb{R}^{M_t}\rightarrow\mathbb{R}^{C_t}$ of task t ($t=1,2,\dots$) defined on datasets $D_t=\{(X_t,Y_t)\}$, we suppose a neural unit has been added $L-1$ nodes one after another directly connected to its readout layer. During each training session t, the output function of the existing network is given by $f_{L-1}(X_t)=\sum_{j=1}^{L-1}\beta_j(t)g_j(X_tw_j+b_j)$ ($L=1,2,\dots,f_0=0$) and the current residual error for training is denoted as $e_{L-1}(t)=f-f_{L-1}$, where w_j and b_j are the hidden parameters, and β_j is the output weights. Then, the following Proposition 4.1 provides a solution to connect the newly generated node g_L (w_L and b_L) to the existing network f_{L-1} when $C_t=1$.

Proposition 4.1. *(Kwok & Yeung, 1997) Let Γ be a set of basis functions g. For a fixed $g\in\Gamma$ ($\|g\|\neq 0$), the expression $\|f-(f_{L-1}+\beta_L g_L)\|$ achieve its minimum iff*

$$\beta_L=\langle e_{L-1},g_L\rangle/\|g_L\|^2 \tag{3}$$

Proposition 4.1 lays the groundwork for how to connect a new node to the existing network but is *limited to the regression problem in single-task learning*. Meanwhile, constructing such a new node by Eq. (3) is less practical in the sense that the reduction of residual error per node expansion will be close to zero (Igelnik & Pao, 1995; Wang & Li, 2017), failing to guarantee preferable learning performance with considerable confidence and convergence rate. The main cause is that fixed random weights, which only perform the forward pass without the backward pass, are prone to incur numerous redundant nodes. Therefore, alternatively, random node generation should be "supervisory" by exerting additional conditions in the forward pass. To this end, we introduce a supervisory mechanism to guide the node generation in modeling neural units for CIL.

In AutoActivator, the former layers are built under the guidance of supervisory mechanism where each layer is made up of a certain number of scalable neural units, while the final classifier layer is step-wise updated by close-formed solutions. The method randomly allocates a batch of nodes in each training session. The batch that causes the highest reduction in training error is added to the neural unit of the current task. For simplicity, Theorem 4.2 formulates neural units over a two-layer AutoActivator, which can be stacked or injected into existing backbones.

Theorem 4.2. *(Universal Approximation Theorem for Convergence Property) Suppose that span(Γ) is dense in L_2 space and $\forall g\in\Gamma$, G is a collection of some g with nonlinear activation. Given $0<r(t)<1$ and a non-negative real number sequence $\{\mu_L(t)\}$ w.r.t. task t ($t=1,2,\dots;c=1,\dots,C_t$), with $\lim_{L\rightarrow+\infty}\mu_L(t)=0$ and $\mu_L(t)\leq(1-r(t))$. For L[2] $=1,2,\dots,$ and step size $l\in\mathbb{N}^{+}$, denoted by*

$$\delta^{\star}_{L}(t)=\sum_{c=1}^{C_t}\delta^{\star}_{L,c}(t),\delta^{\star}_{L,c}(t)=(1-r(t)-\mu_L(t))\|e^{\star}_{L-l,c}(t)\|^2 \tag{4}$$

If a batch of new nodes $G_l(X_tW_l+B_l)$[3] *are randomly generated to satisfy the following inequalities:*

$$\langle e^{\star}_{L-l,c}(t),G_l\beta_{l,c}(t)\rangle\geq\delta^{\star}_{L,c}(t),\ c=1,2,\dots,C_t \tag{5}$$

and connected to the existing neural unit through the output weights in the following least-squares sense:

$$[\beta^{\star}_1(t),\dots,\beta^{\star}_L(t)]=\arg\min_{\beta(t)}\|f-\sum_{j=1}^{L}\beta_j(t)g_j\| \tag{6}$$

Then, we have $\lim_{L\rightarrow+\infty}\|f-f^{\star}_L\|=0$, where $f^{\star}_L=f^{\star}_{L-l}+\beta^{\star}_l(t)G_l$ and $G_l=[g_{L-l+1},\dots,g_L]$.

Proof. See Appendix A. □

Based on Theorem 4.2, we can formulate the indicator $\xi_L(t)=\sum_{c=1}^{C_t}\xi_{L,c}(t)$ of supervisory mechanisms for guiding node expansion by a designated step size l, among which

$$\begin{aligned}\xi_{L,c}(t)=&\langle e^{\star}_{L-l,c}(t),G_l\beta_{l,c}(t)\rangle\\&-(1-r(t)-\mu_L(t))\langle e^{\star}_{L-l,c}(t),e^{\star}_{L-l,c}(t)\rangle>0\end{aligned} \tag{7}$$

Intuitively, $0<r(t)<1$ matters in the residual error decreasing speed of task t, which resembles the learning rate

[2]Without loss of generality, we refer to $L=L(t)$ for simplicity.

[3]Note that $G_l=g_l$ in the special case of $l=1$.

of gradient descent but differs in its explicit scope; $\mu_L(t)$ can be seen as the balance coefficient for ensuring the convergence of Theorem 4.2, which can be found in its proof.

Remark 4.3. The theoretical analysis redefines Eq. (3) *in the context of classification problem for learning sequential tasks*. Modeling neural units with supervisory mechanisms can easily add additional capacity. For one thing, it guarantees the convergence property of a model on a sequence of tasks. In particular, we generate several candidate nodes satisfying supervisory mechanisms in a one-by-one or batch-by-batch manner simultaneously and only recruit ones that cause a maximum reduction in residual errors. This is beneficial to accelerate the convergence rate. For another, during sequential training, the required expansion quota potentially matches the inherent difficulty of each newly arriving task. This not only constructs a near-minimal network architecture for CIL but also suffices to train more incoming tasks.

4.2. Reactivating Updated Neural Units

This section addresses the aforementioned second question, i.e., update the connection weights between neural units and the readout layer (termed output weights) and then reactivate the involved neural units as activators for decision-making. Unlike neural networks with an empirically fixed topology that is unitedly trained, the updates of output weights under node expansion must be repeated every time nodes are added. Hence, computational efficiency becomes the main bottleneck when leveraging the commonly used back-propagation algorithm.

Update. Here we seek an alternative solution under the premise of supervisory mechanisms, which ensures that the nodes already existing in each neural unit are indispensable in learning a given task. That is the pseudoinverse of a partitioned matrix described by some earlier studies (Leonides, 2012; Ben-Israel & Greville, 2003; Chen & Wan, 1999). A dynamic stepwise updating algorithm is then used to update the output weights instantly. It applies to the node expansion process in Theorem 4.2, as illustrated in the following.

Assume that there has been a matrix A_L and we expand it by adding additional l ($l = 1, 2, \dots$) node(s). Let G_l be the resulting matrix. For task t, outputs of the neural unit are given by $\hat{Y}_t = A_L W_L^\star(t)$, where $W_L^\star(t) = [\beta_1^\star(t), \dots, \beta_L^\star(t)]$ as presented in Eq. (6). Denote by $A_{L+l} = [A_L, G_l]$, we have

$$W_{L+l}^\star(t) \triangleq (A_{L+1})^\dagger Y_t = \begin{bmatrix} W_L^\star(t) - DB^{\mathrm{T}}Y_t \\ B^{\mathrm{T}}Y_t \end{bmatrix} \quad (8)$$

where the pseudoinverse of A_{L+l} is

$$(A_{L+1})^\dagger = \begin{bmatrix} (A_L)^\dagger - DB^{\mathrm{T}} \\ B^{\mathrm{T}} \end{bmatrix} \quad (9)$$

$$B^{\mathrm{T}} = \begin{cases} C^\dagger & \text{if } C \neq 0 \\ (D^{\mathrm{T}}D + I)^{-1}D^{\mathrm{T}}(A_L)^+ & \text{if } C = 0 \end{cases} \quad (10)$$

where $C = G_l - A_L D$ and $D = (A_L)^\dagger G_l$.

This way, updating output weights has three strengths. First, the update process can be easily finished *without complete retraining*, i.e., a new pseudoinverse matrix is obtained through a simple calculation of the existing ones. Second, it updates the output weights progressively when adding nodes to each neural unit. In each step, the output weights are theoretically the optimal solution in the least-square sense. Third, only output weights need to be optimized every time, as explicitly expressed in close-formed solutions. Therefore, it avoids the back-propagation of error signals.

Reactivation. In the CIL scenario, task identities are typically absent at the inference time. Hence, another key technical point of AutoActivator is to partially or entirely reactivate the required neural units for task-agnostic decision-making. Herein an *activation threshold* is calculated during training to be used for activating some of the neural units at the test time. Suppose T tasks with $\sum_{t=1}^{T} C_t$ classes have been sequentially trained, yielding T neural units. The following explains how our approach automatically performs the reactivation when fed test instances from tasks 1 to T.

We first retrospect the training phase to passingly introduce the activation threshold of each neural unit, averaged by the highest predicted probability belonging to class c ($c = 1, 2, \dots, C_t$):

$$threshold(t) = mean.\max.(softmax(\hat{Y}_t) - \alpha(t)) \quad (11)$$

where $\alpha(t) = 1/C_t$ is the lowest probability triggering a decision. Hence, a neural unit is finally parameterized by its supervisory mechanism-based random weights (and biases), dynamically stepwise updated output weights, and the activation thresholds. Then, for test instances of any classes seen so far, the objective is to minimize the distance:

$$c \leftarrow \arg\min_t |threshold(t) - \tilde{threshold}(t)| \quad (12)$$

where "$\leftarrow$" means returning index, c is the predicted class, and $\tilde{threshold}(t)$ is computed in Eq. (11) but with test instances. Indeed, Eq. (11) rectifies the inter-class confusion occurring in the similar $softmax$ outputs given test instances from different tasks; Eq. (12) further brings these results to a comparable level and returns the predicted class by tracking the minimal distance in terms of the activation thresholds. From the perspective of methodology, it works on test images arriving in batches or instances, among which feeding batches of test instances from a certain task or class trained facilitates the reactivation progress. Therefore, our AutoActivator can selectively reactivate the involved neural units to retrieve knowledge and leave the remaining inactivated to prevent interference. We summarize its CIL procedure in Algorithm 1 in Appendix B.

Table 1. Comparison results on the split MNIST and FashionMNIST datasets with MLP. We report the average classification accuracy (ACA), backward transfer (BWT) across five random task-order runs, and the aligned memory budget (MB) = model size (#model, MB) + exemplar buffer (#exemplar, MB), where 0 means no rehearsal. No pre-training is used for AutoActivator and all the compared methods.

METHOD	MEMORY BUDGET		MNIST-10/5		FASHIONMNIST-10/5	
	#MODEL ↓	#EXEMPLAR ↓	ACA(%) ↑	BWT ↑	ACA(%) ↑	BWT ↑
SGD (LOWER BOUND)	1.82	-	∼19.91	-	∼19.81	-
MTL (UPER BOUND)	1.82	-	∼98.56	-	∼96.61	-
MAS	5.47	0	44.61±6.62	−0.06±0.06	34.91±5.47	−0.61±0.09
OLEWC	5.47	0	57.38±4.04	−0.38±0.06	54.09±4.03	−0.40±0.08
SI	5.47	0	69.44±4.37	−0.04±0.09	52.11±2.22	−0.49±0.06
EFT	16.54	0	82.53±1.15	−0.09±0.07	74.79±1.23	−0.13±0.05
PCL	3.01	0	94.14±0.67	−0.03±0.03	83.27±0.81	−0.12±0.01
AOP	5.39	0	94.43±0.21	−0.05±0.02	82.97±0.95	−0.14±0.04
CRNET	**1.81**	0	94.49±0.32	−0.02±0.02	86.01±0.74	−0.09±0.01
FS-DGPM	1.82	5.98	89.12±1.14	−0.08±0.01	80.89±0.74	−0.12±0.02
NISPA	1.82	5.98	91.07±0.86	−0.04±0.00	80.93±0.59	−0.15±0.02
BIC	1.82	5.98	93.93±0.58	−0.04±0.01	82.36±0.72	−0.11±0.03
ARI	3.65	5.98	93.60±0.57	−0.04±0.00	82.89±0.83	−0.10±0.01
CO^{2}L	1.82	5.98	93.78±0.24	−0.04±0.00	80.93±0.59	−0.15±0.02
RPS-NET	14.60	5.98	94.53±1.92	−0.02±0.01	84.18±1.60	**−0.03**±0.01
LOGD	1.82	5.98	94.87±0.59	−0.04±0.01	84.39±0.47	−0.09±0.03
IL2M	1.82	5.98	95.51±0.42	−0.04±0.01	82.38±2.04	−0.15±0.03
AUTOACTIVATOR (OURS)	2.04	**0**	**97.32**±0.03	**0.00**±0.00	**88.46**±0.06	-0.09±0.08

5. Experiment

5.1. Experiment Setting

Datasets. We experiment on multiple datasets commonly used for CIL. **Small Scale:** Both MNIST (LeCun et al., 1998) and FashionMNIST (Xiao et al., 2017) are respectively split into 5 disjoint tasks with 2 classes per task. The evaluation starts with the toy examples since AutoActivator belongs to the theoretical and embryonic approach. **Medium Scale:** CIFAR-100 (Krizhevsky et al., 2009) is divided into 10 (25) tasks with each task containing 10 (4) disjoint classes. **Large Scale:** ImageNet-R (Hendrycks et al., 2021) has 200 classes with 24,000 samples for training and 6,000 for testing. It is split into 10 tasks with 20 classes in each task. The substantial intra-class variability renders it more akin to intricate real-world problems. We provide details of the data splits used in Appendix C.1.

Baselines. We extensively compare our method with (i) representative and the latest CIL baselines: **IL2M** (Belouadah & Popescu, 2019), **BiC** (Wu et al., 2019), **LOGD** (Tang et al., 2021), **FS-DGPM** (Deng et al., 2021), **Co^{2}L** (Cha et al., 2021), **ARI** (Wang et al., 2022b), **NISPA** (Gurbuz & Dovrolis, 2022), **DDGR** (Gao & Liu, 2023), **SI** (Zenke et al., 2017), **MAS** (Aljundi et al., 2018), **OLEWC** (Schwarz et al., 2018), **AOP** (Guo et al., 2022), **CRNet** (Li & Zeng, 2023), **RPS-Net** (Rajasegaran et al., 2019), **EFT** (Verma et al., 2021), **DER** (Yan et al., 2021), **PCL** (Hu et al., 2021), **MORE** (Kim et al., 2022), **CLDNet** (Li et al., 2024b), among which the original FS-DGPM uses a multi-head incremental setting where each task has a separate classifier but task identities are not provided in our experiments; (ii) recent prompt-based methods over the pre-trained ViT: **L2P** (Wang et al., 2022d), **DualPrompt** (Wang et al., 2022c), **CODA-Prompt** (Smith et al., 2023); and (iii) non-CIL baselines: stochastic gradient descent for sequential training (**SGD**; approximate lower bound) and joint multiple-task learning (**MTL**; approximate upper bound).

Training. We refer to every aforementioned method's original codebases for implementation and hyper-parameter selection to ensure the best performance. We repeat each experiment five times with randomly shuffled task orderings to get the mean and the standard deviation estimates. More implementation details for *architectures*, *hyper-parameters*, and *metrics* are provided in Appendix C.

5.2. Main Comparison Results

MNIST-10/5 and FashionMNIST-10/5. Table 1 extensively compares different baselines on two standard benchmark datasets adapted for CIL. Our AutoActivator provides strong CIL performance with respect to three measurements. (i) ACA: it achieves the best average accuracy of 97.32% and 88.46%, surpassing the second-best competitors by 1.81% and 2.45%, respectively; (ii) BWT: it behaves with almost zero forgetting during sequentially learning five 2-class tasks, like some work in the task-incremental learning (TIL) scenario where task identities are required to match specific masks (Kang et al., 2022); and (iii) Memory budget: our final #model is slightly (∼12%) larger than the rehearsal-based ones but our method needs no #exemplar buffers.

Table 2. Comparison results on the split CIFAR-100 dataset with ResNet-18, including MORE that originally uses DeiT-S/16 (Touvron et al., 2021). Scale ratio approximately gives the % of the final memory budget (MB) and the initial model size (MB), averaged over the two sequences. All methods allow to start with the same pre-training or learning from scratch and only the winner results are reported.

METHOD	SCALE RATIO (%) ↓	CIFAR-100/10 ACA(%) ↑	CIFAR-100/10 BWT ↑	CIFAR-100/25 ACA(%) ↑	CIFAR-100/25 BWT ↑
SGD	-	7.75±0.17	-	3.18 ±0.23	-
NISPA	>200	37.60±0.73	−0.25±0.02	29.50±0.66	−0.31±0.01
LOGD	119.26	47.45±0.31	−0.13±0.03	48.71±0.23	−0.16±0.00
RPS-NET	>200	58.95±0.25	−0.18±0.01	57.43±0.35	−0.19±0.01
DDGR	>200	59.84±0.57	-	59.15±0.63	-
IL2M	119.26	60.14±0.68	−0.10±0.01	61.33±0.32	**−0.02**±0.02
BIC	155.34	61.03±0.71	−0.09±0.00	60.24±0.59	−0.08±0.04
PCL	146.02	63.58±0.37	−0.11±0.01	62.84±0.43	−0.12±0.01
Co^{2}L	>200	64.31±0.47	−0.15±0.02	63.67±0.28	−0.11±0.01
DER	>200	65.29±1.01	−0.16±0.01	63.54±0.97	−0.18±0.01
CLDNET	115.82	65.42±0.36	-	64.98±0.42	-
MORE	120.08	67.13±1.03	-	66.95±0.98	-
FS-DGPM	191.71	67.54±0.36	−0.23±0.02	68.45±0.38	−0.27±0.04
OURS	**114.26**	**69.65**±0.14	**−0.01**±0.01	**70.16**±0.20	−0.03±0.01

Table 3. Comparison results on the split ImageNet-R dataset using pre-trained ViT. Buffer counts the number of exemplars saved for rehearsal. Forgetting (denoted by $\mathcal{F}$) is negatively correlated with BWT. All results except ours and CODA-Prompt (Smith et al., 2023) are extracted from (Wang et al., 2022c). Note that the original CODA-Prompt uses an easier accuracy-related AIA metric than others and we have aligned it here.

METHOD	BUFFER ↓	ACA(%) ↑	$\mathcal{F}$ ↓
ER	1 000	55.13±1.29	0.35±0.52
BIC		52.14±1.08	0.37±1.05
GDUMB		38.32±0.55	-
DER++		55.47±1.31	0.35±1.50
Co^{2}L		53.45±1.55	0.37±1.81
L2P	0	61.57±0.66	0.10±0.20
DUALPROMPT		68.13±0.49	0.05±0.20
CODA-PROMPT		69.01±0.55	0.05±0.20
OURS†		65.45±0.85	0.03±0.20
OURS††		**70.32**±0.55	**0.02**±0.20
UPPER BOUND	-	79.13±0.18	-

These results indicate that AutoActivator well trades off between model accuracy and memory budget. This makes sense for real-world applications under privacy-sensitive and resource-limited CIL scenarios. Meanwhile, the steady standard deviations show that our method has strong task-order robustness, with similar results regardless of random orderings for 5 independent runs.

CIFAR-100/10 and CIFAR-100/25. Table 2 reports two more challenging task sequences evenly split by the widely-used visual benchmark CIFAR-100. Based on the empirical evaluation, our method achieves the best average classification accuracy of 69.65% and 70.16%, improving upon the second-best method by 2.11% on CIFAR-100/10 and 1.71% on CIFAR-100/25, respectively. Interestingly, although pre-training implicitly alleviates the effects of catastrophic forgetting, it is not necessarily translated to CIL performance. We observe that some well-known methods still suffer from a large forgetting given the pre-trained backbone compared with a learning-from-scratch paradigm, similar findings can be observed in (Wang et al., 2022c). In addition to average classification accuracy, our method outperforms the selected state-of-the-art methods on scale ratio, indicating that the network expansion is compactly commensurate with the intrinsic complexity of a task sequence. Again, the proposed method yields competitive standard deviations indicating the superiority task-order robustness.

ImageNet-R-200/10. Table 3 records the performance of compared methods starting from the same ImageNet pre-trained ViT-B/16 for the split ImageNet-R sequence. This yields two versions of our system: Ours† refers to only one pre-trained ViT being used; Ours†† refers to attaching complementary prompts to two pre-trained ViT, resembling what DualPrompt learns two sets of disjoint prompt spaces but are untouched during CIL process. We observe that it is still particularly challenging for rehearsal-based methods with a buffer size of 1 000. By contrast, there is an obvious gain for the recent emerging prompt-based methods, even though they are rehearsal-free. It is worth mentioning that Ours†† outperforms the strongest competitors DualPrompt and CODA-Prompt by 2.19% and 1.31%, respectively. This implies that prompt-based methods have not come close to exhausting their capacity for accuracy, e.g., combining with our method for a marginal boost. The proposed connectionist model, which tailors neural unit dynamics with its convergence property theoretically guaranteed, can thus serve as a general CIL classifier. Meanwhile, our method shows competitive performance on the metric Forgetting. In

Table 4. Parameter analysis of supervisory mechanisms. We report the ACA, the cumulative number of nodes (Nodes), and the whole running time (Time) per task sequence under different l and T_{max}.

STEP SIZE	T_{max}	MNIST-10/5			FASHIONMNIST-10/5		
		ACA(%)↑	NODES↓	TIME(S)↓	ACA(%)↑	NODES↓	TIME(S)↓
$l = 1$	1	96.86	692	12.87	88.22	702	13.13
	10	97.08	640	18.38	88.32	623	16.26
	50	97.30	586	28.28	88.54	574	30.50
	200	97.31	532	62.45	88.53	512	48.99
$l = 10$	1	96.82	700	6.76	88.19	720	6.83
	10	97.05	680	10.92	88.37	680	13.39
	50	97.24	670	19.74	88.51	660	17.95
	200	97.21	650	42.83	88.52	650	33.40

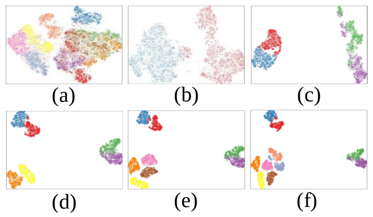

Figure 2. t-SNE visualization where each color represents a class. (a) Mixed raw sample space based on the training data of ten classes as a reference. (b)-(f) Well-clustered representation space based on neural units' outputs after learning two classes per session.

Table 5. Influence of expected accuracy in supervisory mechanisms. We report the neural unit size per task ($t = 1, 2, \dots, 5$) and the average number of parameters per class.

$R(t)$	1	2	3	4	5	#PARAM (M)
99%	200	200	50	30	200	0.0535
98%	200	200	20	20	100	0.0425
95%	20	70	20	10	20	0.0110

Table 6. Effectiveness of reactivation process, measured by the ACA(%) under different components.

COMPONENT I	✗	✓	✓
COMPONENT II	✗	✗	✓
MNIST-10/5	13.69	97.18	**97.21**
FASHIONMNIST-10/5	22.05	86.46	**88.37**
CIFAR-100/10	5.56	69.21	**69.27**
IMAGENET-R	4.98	63.25	**65.83**

a nutshell, prompt-based methods exhibit a wise utilization of pre-trained transformer-based backbones while our AutoActivator seeks an effective and scalable implementation.

5.3. Analysis of Modifying Neural Unit Dynamics

We now conduct an extensive empirical investigation to display the effectiveness of modules in the proposed novel connectionist model. This also covers the ablation study and parameter analysis, which pertains to each component serving for CIL. We first investigate two key components in supervisory mechanisms through the lens of CIL: (1) step size l for recruiting node(s) each time—we refer to $l = 1$ as a one-by-one version and $l \geq 2$ as a batch-by-batch version; and (2) the maximum times of random generation T_{max}—it determines the number of attempts for mining candidate nodes that meet with a supervisory mechanism.

One-by-One v.s. Batch-by-Batch Version. It can be observed from Table 4 that both versions could achieve the same-level results. However, under the equal T_{max} setting, the one-by-one version ($l = 1$) shows superiority in expanding more compact neural units as adding only one node each time is potentially more targeted; By contrast, the batch-by-batch version ($l = 10$) could significantly reduce the time requirements. It is worth mentioning that the results of both versions under proper T_{max} (e.g., $T_{max} = 50$ used in our experiments) are at an acceptable level.

The Maximum Times of Random Generation. Also from Table 4, we observe that a larger T_{max} contributes to shrinking the Nodes while slightly improving the ACA without sacrificing the Time too much. However, an excessively high value would make the construction of the candidate pool time-consuming, while a too little one would give rise to learning instability for failure in finding nodes that are useful enough. Hence, T_{max} is related to both opportunity and efficiency in the process of node generation.

Representation Learning of Neural Units. We also visualize the representation learning of scalable neural units. Fig. 2 depicts the t-SNE visualization (Van der Maaten & Hinton, 2008) of the raw sample space and the output representations of neural units. We observe that the same classes are well clustered while different classes are properly separated. Therefore, the representation learning of neural units could provide useful information for decision-making, i.e., to adjust the decision boundary for all classes simultaneously, and facilitate the dynamic stepwise update of output weights in the least-square sense.

The Actual Expansion Quota. To indicate the expansion quota on given problems, we gather the neural unit size and count the number of parameters (#param). Taking FashionMNIST-10/5 as an example, we specify the maximum number of nodes $L_{max}(t) = 200$ as one of the termination criteria and vary another termed expected accuracy $R(t)$ for node expansion. Table 5 shows that the final expansion quota is commensurate with the intrinsic complexity of every task, e.g., only 30 nodes are recruited for Task 4 under $R(t) = 99\%$. Meanwhile, the node expansion slightly introduced additional parameters, about 0.0535 M for each class, which only equals the level of replaying 7 exemplars.

Ablation Study of Activation Thresholds. We then validate the effectiveness of reactivation for defying the inter-class confusion in CIL. Note that Component I by Eq. (11) rectifies inter-class confusion and Component II by Eq. (12) calibrates the results. Table 6 shows that our model would suffer serious forgetting without them. By contrast, using Component I could tackle this issue in most cases, together with Component II for a marginal boost. In particular, the latter works well for the case of substantial intra-class variability, demonstrating that AutoActivator could exactly reactivate the involved neural units without recourse to task identities. More insights and analysis are given in Appendix D.

6. Conclusion

We propose to harness neural unit dynamics for efficient and scalable CIL. Unlike most architecture-based methods whose expansion criterion relies on changes of the loss, which lacks theoretical guarantees, the supervisory mechanism narrows the gap using a universal approximation theorem. The reactivation paradigm pioneered is biologically plausible, and we believe this has a lot of potential room for exploration. Sufficient comparison experiments and ablation analysis display the effectiveness of our model. Other interesting investigations that we leave for future work include combining our approach with class-imbalance sequences, which may benefit from an AutoActivator-like algorithm.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grants 623B2040 and 61936004, and the 111 Project on Computational Intelligence and Intelligent Control under Grant B18024.

Impact Statement

This paper presents work whose goal is to advance the field of General Machine Learning. We would like to discuss the potential societal consequences that may result from the limitation of our work. The supervised learning dataset we used by following most existing CIL methods involves the availability of a task boundary during training. From a practical perspective, however, it can be a restrictive assumption and somewhat not be real-world applicable. We leave further study (e.g., novel class discovery technique) on this as future work. Besides, we assume the availability of a pre-trained backbone. While this assumption is widely accepted, as pre-trained models have become a common asset in advanced vision communities, it is important to consider that biases and fairness issues present in the original model may persist during the CIL process. To mitigate the bias and fairness issues, we strongly recommend that users conduct a thorough examination of the pre-trained models.

References

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision*, pp. 139–154, 2018.

Bang, J., Kim, H., Yoo, Y., Ha, J.-W., and Choi, J. Rainbow memory: Continual learning with a memory of diverse samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8218–8227, 2021.

Belouadah, E. and Popescu, A. IL2M: Class incremental learning with dual memory. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 583–592, 2019.

Ben-Israel, A. and Greville, T. N. *Generalized Inverses: Theory and Applications*, volume 15. Springer Science & Business Media, 2003.

Bonicelli, L., Boschini, M., Porrello, A., Concetto, S., Calderara, S., et al. On the effectiveness of lipschitz-driven rehearsal in continual learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 31886–31901, 2022.

Cha, H., Lee, J., and Shin, J. Co^{2}L: Contrastive continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9516–9525, 2021.

Chen, C. P. and Wan, J. Z. A rapid learning and dynamic stepwise updating algorithm for flat neural networks and the application to time-series prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(1):62–72, 1999.

Deng, D., Chen, G., Hao, J., Wang, Q., and Heng, P.-A. Flattening sharpness for dynamic gradient projection memory benefits continual learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 18710–18721, 2021.

Douillard, A., Cord, M., Ollion, C., Robert, T., and Valle, E. PODNet: Pooled outputs distillation for small-tasks incremental learning. In *Computer vision–ECCV 2020: 16th European conference, Glasgow, UK, August 23–28, 2020, proceedings, part XX 16*, pp. 86–102. Springer, 2020.

Gao, R. and Liu, W. DDGR: continual learning with deep diffusion-based generative replay. In *International Conference on Machine Learning*, pp. 10744–10763. PMLR, 2023.

Guo, Y., Hu, W., Zhao, D., and Liu, B. Adaptive orthogonal projection for batch and online continual learning. In

Proceedings of the AAAI Conference on Artificial Intelligence, pp. 6783–6791, 2022.

Guo, Y., Liu, B., and Zhao, D. Dealing with cross-task class discrimination in online continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11878–11887, 2023.

Gurbuz, M. B. and Dovrolis, C. NISPA: Neuro-inspired stability-plasticity adaptation for continual learning in sparse networks. In *International Conference on Machine Learning*, pp. 8157–8174. PMLR, 2022.

Hayes, T. L., Kafle, K., Shrestha, R., Acharya, M., and Kanan, C. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pp. 466–483. Springer, 2020.

Hendrycks, D., Basart, S., Mu, N., Kadavath, S., Wang, F., Dorundo, E., Desai, R., Zhu, T., Parajuli, S., Guo, M., et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8340–8349, 2021.

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Hu, W., Qin, Q., Wang, M., Ma, J., and Liu, B. Continual learning by using information of each class holistically. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7797–7805, 2021.

Hu, Z., Li, Y., Lyu, J., Gao, D., and Vasconcelos, N. Dense network expansion for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11858–11867, 2023.

Huang, C., Li, M., Cao, F., Fujita, H., Li, Z., and Wu, X. Are graph convolutional networks with random weights feasible? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):2751–2768, 2023.

Igelnik, B. and Pao, Y.-H. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6(6):1320–1329, 1995.

Joseph, O., Pleydell-Bouverie, B., Dupret, D., and Csicsvari, J. Play it again: reactivation of waking experience and memory. *Trends in Neurosciences*, 33(5):220–229, 2010.

Kang, H., Mina, R. J. L., Madjid, S. R. H., Yoon, J., Hasegawa-Johnson, M., Hwang, S. J., and Yoo, C. D. Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pp. 10734–10750. PMLR, 2022.

Ke, Z., Liu, B., Ma, N., Xu, H., and Shu, L. Achieving forgetting prevention and knowledge transfer in continual learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 22443–22456, 2021.

Kim, G., Liu, B., and Ke, Z. A multi-head model for continual learning via out-of-distribution replay. In *Conference on Lifelong Learning Agents*, pp. 548–563. PMLR, 2022.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Handbook of Systemic Autoimmune Diseases*, 2009.

Kwok, T.-Y. and Yeung, D.-Y. Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, 8(5):1131–1148, 1997.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Leonides, C. *Control and Dynamic Systems V18: Advances in Theory and Applications*. Elsevier, 2012.

Li, D. and Zeng, Z. CRNet: A fast continual learning framework with random theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10731–10744, 2023.

Li, D., Wang, T., Chen, J., Kawaguchi, K., Lian, C., and Zeng, Z. Multi-view class incremental learning. *Information Fusion*, 102:102021, 2024a.

Li, D., Wang, T., Chen, J., Ren, Q., Kawaguchi, K., and Zeng, Z. Towards continual learning desiderata via hsic-bottleneck orthogonalization and equiangular embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 13464–13473, 2024b.

Li, M. and Wang, D. Insights into randomized algorithms for neural networks: Practical issues and common pitfalls. *Information Sciences*, 382:170–178, 2017.

Lin, H., Zhang, B., Feng, S., Li, X., and Ye, Y. PCR: Proxy-based contrastive replay for online class-incremental continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24246–24255, 2023.

Liu, Y., Schiele, B., and Sun, Q. Adaptive aggregation networks for class-incremental learning. In *Proceedings* *of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 2544–2553, 2021.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 6470–6479, 2017.

Márton, C. D., Zhou, S., and Rajan, K. Linking task structure and neural network dynamics. *Nature Neuroscience*, 25(6):679–681, 2022.

Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A. D., and van de Weijer, J. Class-incremental learning: survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5513–5533, 2023.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Elsevier, 1989.

McDonnell, M. D., Gong, D., Parvaneh, A., Abbasnejad, E., and van den Hengel, A. RanPAC: Random projections and pre-trained models for continual learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mehta, S. V., Patil, D., Chandar, S., and Strubell, E. An empirical investigation of the role of pre-training in lifelong learning. *Journal of Machine Learning Research*, 24(214):1–50, 2023.

Pao, Y.-H. and Takefuji, Y. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

Qiao, Z., Pham, Q., Cao, Z., Le, H. H., Suganthan, P. N., Jiang, X., and Savitha, R. Class-incremental learning for time series: Benchmark and evaluation. *arXiv preprint arXiv:2402.12035*, 2024.

Rajasegaran, J., Hayat, M., Khan, S., Khan, F. S., and Shao, L. Random path selection for incremental learning. In *Advances in Neural Information Processing Systems*, volume 32, pp. 12669–12679, 2019.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020.

Rasch, B. and Born, J. Maintaining memories by reactivation. *Current Opinion in Neurobiology*, 17(6):698–703, 2007.

Rios, A., Ahuja, N., Ndiour, I., Genc, U., Itti, L., and Tickoo, O. incDFM: Incremental deep feature modeling for continual novelty detection. In *European Conference on Computer Vision*, pp. 588–604. Springer, 2022.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537, 2018.

Serrà, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Shim, D., Mai, Z., Jeong, J., Sanner, S., Kim, H., and Jang, J. Online class-incremental continual learning with adversarial shapley value. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 9630–9638, 2021.

Smith, J. S., Karlinsky, L., Gutta, V., Cascante-Bonilla, P., Kim, D., Arbelle, A., Panda, R., Feris, R., and Kira, Z. CODA-Prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11909–11919, 2023.

Tang, S., Chen, D., Zhu, J., Yu, S., and Ouyang, W. Layerwise optimization by gradient decomposition for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9634–9643, 2021.

Tang, Y.-M., Peng, Y.-X., and Zheng, W.-S. When prompt-based incremental learning does not meet strong pretraining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1706–1716, 2023.

Tong, S., Dai, X., Wu, Z., Li, M., Yi, B., and Ma, Y. Incremental learning of structured memory via closed-loop transcription. In *International Conference on Learning Representations*, 2023.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

Vahedian, F., Li, R., Trivedi, P., Jin, D., and Koutra, D. Convolutional neural network dynamics: A graph perspective. *arXiv preprint arXiv:2111.05410*, 2021.

Van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):4069, 2020.

Van der Maaten, L. and Hinton, G. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

Verma, V. K., Liang, K. J., Mehta, N., Rai, P., and Carin, L. Efficient feature transformations for discriminative and generative continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13865–13875, 2021.

Vyas, S., Golub, M. D., Sussillo, D., and Shenoy, K. V. Computation through neural population dynamics. *Annual Review of Neuroscience*, 43:249–275, 2020.

Wang, D. and Li, M. Stochastic configuration networks: Fundamentals and algorithms. *IEEE Transactions on Cybernetics*, 47(10):3466–3479, 2017.

Wang, F.-Y., Zhou, D.-W., Ye, H.-J., and Zhan, D.-C. FOSTER: Feature boosting and compression for class-incremental learning. In *European Conference on Computer Vision*, pp. 398–414. Springer, 2022a.

Wang, F.-Y., Zhou, D.-W., Liu, L., Ye, H.-J., Bian, Y., Zhan, D.-C., and Zhao, P. BEEF: Bi-compatible class-incremental learning via energy-based expansion and fusion. In *International Conference on Learning Representations*, 2023.

Wang, R., Bao, Y., Zhang, B., Liu, J., Zhu, W., and Guo, G. Anti-retroactive interference for lifelong learning. In *European Conference on Computer Vision*, pp. 163–178. Springer, 2022b.

Wang, Z., Zhang, Z., Ebrahimi, S., Sun, R., Zhang, H., Lee, C.-Y., Ren, X., Su, G., Perot, V., Dy, J., et al. DualPrompt: Complementary prompting for rehearsal-free continual learning. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI*, pp. 631–648. Springer, 2022c.

Wang, Z., Zhang, Z., Lee, C.-Y., Zhang, H., Sun, R., Ren, X., Su, G., Perot, V., Dy, J., and Pfister, T. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022d.

Wei, Y., Ye, J., Huang, Z., Zhang, J., and Shan, H. Online prototype learning for online continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 18764–18774, 2023.

Wołczyk, M., Piczak, K., Wójcik, B., Pustelnik, L., Morawiecki, P., Tabor, J., Trzcinski, T., and Spurek, P. Continual learning with guarantees via weight interval constraints. In *International Conference on Machine Learning*, pp. 23897–23911. PMLR, 2022.

Wu, T.-Y., Swaminathan, G., Li, Z., Ravichandran, A., Vasconcelos, N., Bhotika, R., and Soatto, S. Class-incremental learning with strong pre-trained models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9601–9610, 2022.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., and Fu, Y. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 374–382, 2019.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yan, S., Xie, J., and He, X. DER: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3014–3023, 2021.

Yang, B., Lin, M., Zhang, Y., Liu, B., Liang, X., Ji, R., and Ye, Q. Dynamic support network for few-shot class incremental learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):2945–2951, 2023.

Zeng, G., Chen, Y., Cui, B., and Yu, S. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995, 2017.

Zhang, C., Bengio, S., and Singer, Y. Are all layers created equal? *Journal of Machine Learning Research*, 23:1–28, 2022.

Zhang, G., Wang, L., Kang, G., Chen, L., and Wei, Y. SLAC: Slow learner with classifier alignment for continual learning on a pre-trained model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19148–19158, 2023.

Zhang, L. and Suganthan, P. N. Visual tracking with convolutional random vector functional link network. *IEEE Transactions on Cybernetics*, 47(10):3243–3253, 2016.

Zhou, D.-W., Wang, Q.-W., Ye, H.-J., and Zhan, D.-C. A model or 603 exemplars: Towards memory-efficient class-incremental learning. In *International Conference on Learning Representations*, 2023.

Zhuang, H., Weng, Z., Wei, H., Xie, R., Toh, K.-A., and Lin, Z. ACIL: Analytic class-incremental learning with absolute memorization and privacy protection. In *Advances in Neural Information Processing Systems*, volume 35, pp. 11602–11614, 2022.

A. Proof of Theorem 4.2

Proof. First, we deduce the intermediate values w.r.t. output weights $\beta_l(t)$, the counterpart of Eq. (3) in Proposition 4.1.

$$
\begin{aligned}
\mathcal{L} &= \|f - f_L\|^2 \\
&= \|f - f_{L-l} - (g_{L-l+1}\beta_{L-l+1}(t) + \cdots + g_L\beta_L(t))\|^2 \\
&= \|e_{L-l}(t) - [g_{L-l+1}, \ldots, g_L][\beta_{L-l+1}(t), \ldots, \beta_L(t)]^{\mathrm{T}}\|^2 \\
&= \|e_{L-l}(t) - G_l\beta_l(t)\|^2 \\
&= \sum_{c=1}^{C_t} \langle e_{L-l,c}(t) - G_l\beta_{l,c}(t), e_{L-l,c}(t) - G_l\beta_{l,c}(t)\rangle \\
&= \sum_{c=1}^{C_t} \big(\langle e_{L-l,c}(t), e_{L-l,c}(t)\rangle - 2\langle e_{L-l,c}(t), G_l\beta_{l,c}(t)\rangle + \langle G_l\beta_{l,c}(t), G_l\beta_{l,c}(t)\rangle\big) \\
&= \|e_{L-l}(t)\|^2 - \sum_{c=1}^{C_t} 2\langle e_{L-l,c}(t), G_l\beta_{l,c}(t)\rangle + \sum_{c=1}^{C_t}\langle G_l\beta_{l,c}(t), G_l\beta_{l,c}(t)\rangle
\end{aligned}
\tag{13}
$$

where $G_l = [g_{L-l+1}, \ldots, g_L]$, $\beta_l(t) = [\beta_{L-l+1}(t), \ldots, \beta_L(t)]^{\mathrm{T}}$, and $e_{L-l}(t) = f_L - f_{L-l}$.

Take the derivative of Eq. (13) w.r.t. $\beta_{l,c}(t)$, we have

$$
\frac{\partial \mathcal{L}}{\partial \beta_{l,c}(t)} = -2G_l^{\mathrm{T}} e_{L-l,c}(t) + 2G_l^{\mathrm{T}} G_l \beta_{l,c}(t) \tag{14}
$$

By setting Eq. (14) to zero, we have

$$
\beta_{l,c}(t) = (G_l^{\mathrm{T}} G_l)^{\dagger} G_l^{\mathrm{T}} e_{L-l,c}(t) \tag{15}
$$

Then, denote by $\beta_l(t) = [\beta_{l,1}, \ldots, \beta_{l,C_t}]$ and $e_L(t) = e^{\star}_{L-l}(t) - G_l\beta_l(t)$ the intermediate values. Given optimal results $[\beta^{\star}_1(t), \ldots, \beta^{\star}_L(t)]$ by Eq. (6), let $e^{\star}_L(t) = f - \sum_{j=1}^{L} \beta^{\star}_j(t) g_j$ ($e^{\star}_0(t) = f$). For the progression $\|e^{\star}_L(t)\|^2$, we have

$$
\begin{aligned}
\|e^{\star}_L(t)\|^2 &\leq \|e_L(t)\|^2 \\
&= \langle e^{\star}_{L-l}(t) - G_l\beta_l(t), e^{\star}_{L-l}(t) - G_l\beta_l(t)\rangle \\
&\leq \|e^{\star}_{L-l}(t)\|^2 - \|G_l\beta_l(t)\|^2 \\
&\leq \|e^{\star}_{L-l}(t)\|^2
\end{aligned}
\tag{16}
$$

We note that the progression $\|e^{\star}_L(t)\|^2$ is monotonically decreasing. Using Eqs. (4-5), we can further obtain

$$
\begin{aligned}
&\|e^{\star}_L(t)\|^2 - (r(t) + \mu_L(t))\|e^{\star}_{L-l}(t)\|^2 \\
\leq &\|e_L(t)\|^2 - (r(t) + \mu_L(t))\|e^{\star}_{L-l}(t)\|^2 \\
= &\sum_{c=1}^{C_t} \langle e^{\star}_{L-l,c}(t) - G_l\beta_{l,c}(t), e^{\star}_{L-l,c}(t) - G_l\beta_{l,c}(t)\rangle - \sum_{c=1}^{C_t} (r(t) + \mu_L(t))\langle e^{\star}_{L-l,c}(t), e^{\star}_{L-l,c}(t)\rangle \\
= &\sum_{c=1}^{C_t} \big((1 - r(t) - \mu_L(t))\langle e^{\star}_{L-l,c}(t), e^{\star}_{L-l,c}(t)\rangle - \sum_{c=1}^{C_t} 2\langle e^{\star}_{L-l,c}(t), G_l\beta_{l,c}(t)\rangle + \sum_{c=1}^{C_t}\langle G_l\beta_{l,c}(t), G_l\beta_{l,c}(t)\rangle \\
= &\sum_{c=1}^{C_t} \delta^{\star}_{L,c}(t)\|e^{\star}_{L-l}(t)\|^2 - e^{\star\mathrm{T}}_{L-l,c}(t) G_l (G_l^{\mathrm{T}} G_l)^{\dagger} G_l^{\mathrm{T}} e^{\star}_{L-l,c}(t) \\
\leq &\sum_{c=1}^{C_t} \delta^{\star}_{L,c}(t) - \langle e^{\star}_{L-l,c}(t), G_l\beta_{l,c}(t)\rangle \\
\leq &0
\end{aligned}
\tag{17}
$$

Therefore, we have $\|e^{\star}_L(t)\|^2 \leq (r(t) + \mu_L(t))\|e^{\star}_{L-l}(t)\|^2$. Note that $0 < r(t) < 1$ and $\lim_{L\to+\infty} \mu_L(t) = 0$, i.e., $\lim_{L\to+\infty} \|e^{\star}_L(t)\| = 0$. This completes the proof. □

B. Algorithm description for AutoActivator

To better illustrate our method, the whole procedure of training and testing is provided in Algorithm 1. We now show the expansion process from the perspective of matrix/vector multiplication. Note that nodes added to each scalable neural unit are in a fully-connected manner but different neural units on a certain layer are not fully connected to the next layer (see Figure 1), for example, their weights are stored in a list specific to a certain layer. This is significantly different from a fully-connected (FC) layer or multi-layer perceptron (MLP) in that our classifier layer is additionally parameterized by the activation threshold for expansion-based decision-making, i.e., reactivating the required neural units in different lists.

Algorithm 1 ActoActivator Training and Test Algorithm

```
Input: Datasets $\{D_t\}_{t=1}^{T}$; Termination criteria of expanding hidden unit $t$: the maximum number of nodes $L_{max}(t)$ and expected accuracy $R(t)$
 1: # During Sequential Training
 2: for $t = 1, 2, \dots, T$ do
 3:     while None of the termination criteria is satisfied do
 4:         Recruit randomly generated nodes based on supervisory mechanisms by Eq. (7)
 5:         Update output weights by Eq. (8)
 6:     end while
 7:     Compute activation thresholds by Eq. (11)
 8: end for
 9: # At Inference Time
10: Given batches of test instances from a certain task
11: Reactivate the required hidden units by Eq. (12)
Output: Task identity $t$ and the predicted classes $c$
```

In our algorithm, we compactly expand the network on multiple layers. Among them, the former layers are built under the guidance of supervisory mechanism (Section 4.1) while the final classifier layer is step-wise updated by close-formed solutions (Section 4.2). Importantly, when it comes to our expansion process, the final classifier layer together with at least one (or multiple) former layer(s) is required. This is because the mentioned two different types of layers work together to complete the inference/forward pass in our algorithm.

Without the loss of generality, we take a two-layer AutoActivator on the split MNIST (five 2-classification tasks) as an example. For the first task whose input has 784 dimensions, we start with modeling the neural unit from scratch with step size l nodes (e.g., $l = 10$) that are randomly generated but recruited for expansion under the guidance of supervisory mechanism. When it meets the termination criteria, it yields input weight matrix $W_{in} \in R^{784\times L_1}$ (e.g., $L_1 = 200$) from the former layer and output weight matrix $W_{out} \in R^{L_1\times 2}$ from the classifier layer. These two matrices are respectively saved in two lists corresponding to two different types of layers, which are also bound to an activation threshold (denoted by $threshold(1)$). For the second task whose input has 784 dimensions, similarly, we start with expanding the former layer by progressively recruiting additional l nodes (e.g., $l = 10$). When it meets the termination criteria, it yields the newly added input weight matrix $W'_{in} \in R^{784\times L_2}$ (e.g., $L_2 = 150$) for the former layer and output weight matrix $W'_{out} \in R^{L_2\times 2}$ for the classifier layer. These two matrices are bound to an activation threshold (denoted by $threshold(2)$). The final weight of the former layer is kept in a list $[W_{in}, W'_{in}]$ without row/column-wise concatenation, and the same goes for the final weight of the classifier layer that is kept in a list $[W_{out}, W'_{out}]$. In this case, the final dimension of the weight in that former layer is composed of separate $784 \times L_1$ and $784 \times L_2$, instead of $(784 + 784) \times (L_1 + L_2)$; the final dimension of the weight in that classifier layer is composed of separate $L_1 \times 2$ and $L_2 \times 2$, instead of $(L_1 + L_2) \times (2 + 2)$. Given the input feature vector from either of trained tasks has $m = 784$ dimensions, only the input-output weight matrix pair that achieves its activation threshold serves for making the decision. Thus the size of output vector is 2 dimensional with remaining deactivated. Note that since the size of L_1 and L_2 is adaptively decided by the complexity/difficulty of a given task, which are usually different, the input feature vector (of size m) should be the original input from that task, e.g., $m = 784$ in the task sequence. This learning paradigm is dramatically different from a multi-head classifier . Our theoretical contribution guarantees the model's convergence property on learning sequential mappings. The above also works for multi-layer AutoActivator. Additionally, AutoActivator can be extended to convolution modules or injected into some advanced backbones such as ResNet or ViT, which accommodates the real-world scenario where pre-training is usually involved as an initial step.

C. Additional implementation details

C.1. Data splits and architectures

We run experiments on extensive datasets adapted for CIL under different widely used backbones, which are implemented in PyTorch with NVIDIA RTX 3080-Ti GPUs. For fair comparisons, (i) all methods select the same or similar-sized neural network architectures; (ii) following the settings in (Ke et al., 2021; Hu et al., 2021; Rios et al., 2022; Zeng et al., 2019; Wu et al., 2022), all methods allow to start from the same pre-training unless otherwise specified; and (iii) the data for pre-training can not include in that of CIL, e.g., we manually remove the overlapping classes. The resulting data splits and architectures used in our experiments are shown in Table 7.

Table 7. Details of the data splits and the selected architectures for pre-training and CIL. These are what we exactly used in our experiments of the main text unless otherwise specified.

DATASET	ARCHIRECTURE	DATA SPLIT	
		PRE-TRAINING	CIL
MNIST FASHIONMNIST	MLP	✗	MNIST-10/5 FASHIONMNIST-10/5
CIFAR-100 IMAGENET-R	RESNET-18 VIT-B/16	TINY-IMAGENET IMAGENET	CIFAR-100/10, CIFAR-100/25 IMAGENET-R-200/10

C.2. Hyper-parameter

We carefully reproduce the selected baselines and use the hyper-parameter settings by referring to their original source code. When conducting experiments with different datasets, we keep about 10% of the training data from each task for validation. With regard to baselines, we use the SGD optimizer with an initial learning rate (0.001 for MNIST, FashionMNIST; 0.01 for the remaining) and do much tuning. For the rehearsal-based approaches, we keep a random exemplar set of 2k per task sequence or ∼200 per class by following the similar setting in (Rajasegaran et al., 2019; Hsu et al., 2018). For regularization-based approaches, the penalty coefficient is from set {100, 1 000, 10 000, 100 000}. For architecture-based approaches, we pay attention to their model size after learning all tasks. In our method, the maximum number of nodes $L_{max}(t)$ and expected accuracy $R(t)$ of neural unit t for task t are problem-dependent and not fixed. Instead, we perform the early termination criteria at the level of node expansion instead of epochs (Serrà et al., 2018) or phases (Gurbuz & Dovrolis, 2022), by tracking the lowest value of the residual error achieved so far on the validation set. This way is flexible in determining the most appropriate hyper-parameter settings without over-fitting, e.g., for MNIST-10/5 and FashionMNIST-10/5, we use $L_{max}(t) = 200$ and $R(t) = 99\%$; for CIFAR-100, we use $L_{max}(t) = 1000$ and $R(t) = 90\%$ (CIFAR-100/10), and $L_{max}(t) = 500$ and $R(t) = 80\%$ (CIFAR-100/25). Similarly, we empirically set the step size $l = 10$ for node expansion each time and the maximum times of random generation $T_{max} = 50$ (see Table 4 for more details); $r(t) = 0.9$ and $\mu_L(t) = \frac{1-r(t)}{L+1}$ based on Theorem 4.2.

To ensure a fair comparison, when in the absence of pre-training, every comparison method uses a similar-sized neural network architecture that is fully trainable; when in the presence of pre-training, every comparison method starts from the same pre-trained backbone as a base session. Then, following the settings in (Hu et al., 2021; Zeng et al., 2019; Hayes et al., 2020; Ke et al., 2021; Wang et al., 2022c), we keep the pre-trained model untouched for methods such as PCL and ours or still make it fully trainable for methods that could not learn well with a frozen backbone, as evaluated in (Wang et al., 2022c). That is, all methods allow to start with the same pre-training or learning from scratch and only the winning results are reported.

C.3. Metrics

We evaluate all considered baselines based on the following metrics (higher is better): **Average Classification Accuracy (ACA)**, i.e., ACA^T, measures the test performance of the final model at hand on all T tasks seen so far:

$$\text{ACA} = \frac{1}{T}\sum_{t=1}^{T} R_{T,t} \tag{18}$$

where $R_{T,t}$ is the test accuracy for task t after training on task T; **Average Incremental Accuracy (AIA)** involves the intermediate results as each step/session proceeds, say

$$\text{AIA} = \frac{1}{T}\sum_{t=1}^{T} ACA^t \tag{19}$$

Among them, it is typically $ACA^1 > ACA^2 > \cdots > ACA^t > \cdots > ACA^T$. Therefore, ACA is more challenging than AIA, and our experiments focus more on the former while using the latter for a direct comparison with certain of competitors. **Backward Transfer (BWT)** (Lopez-Paz & Ranzato, 2017) indicates a model's ability in knowledge retention, averaged over all tasks:

$$\text{BWT} = \frac{1}{T-1}\sum_{t=1}^{T-1} R_{T,t} - R_{t,t} \tag{20}$$

Negative BWT values mean that learning new tasks causes forgetting past tasks while a model with BWT = 0 can be considered forgetting-free (Kang et al., 2022).

As an additional metric (fewer is better), we report the **Memory Budget** by aligning the memory cost of network parameters and old samples, i.e., switching them to a 32-bit floating number. In this way, both the final model size (#model, MB) and exemplar buffers (#exemplar, MB) are counted into the memory budget (MB), calculated with an approximate summation of them. For example, the budget for saving a simple 2-layer MLP (with [784-400-400-10] neurons) converts to 478 410 floats$\times$4 bytes/float $\div$ (1 024$\times$1 024) bytes $\approx$ 1.82 MB; the budget for saving 2k samples (with 28×28 gray-scaled pixels) converts to 1 568 000 floats$\times$ 4 bytes/float $\div$ (1 024$\times$1 024) bytes $\approx$ 5.98 MB, as reported in Table 1.

D. Additional Comparison Results

D.1. Experiments on Unevenly Split Task Sequence

Unlike the intra-sequence balanced CIL where a dataset with C classes evenly divided into T tasks, e.g., MNIST-10/5 and CIFAR-100/10, we now preliminarily investigate the *intra-sequence imbalanced CIL* where a dataset with C classes are unevenly split into C_t classes—CIFAR-$\{100(C_t)|C_t \neq C_{t+1}\}$. This yields CIFAR-$\{$100(10), 100(20), 100(30), ...$\}$ and CIFAR-$\{$100(2), 100(4), ...$\}$, in which the value in parentheses indicates the number of classes for the current task. Figure 3 evaluates different algorithms under such a more realistic CIL where tasks are not evenly split. Compared with Table 2, almost all the methods experience performance degradation since the intra-sequence imbalanced case has something to do with the choice of architecture or expansion. This is reflected in the different numbers of both classes and samples within each task. In the resulting 4-task and 10-task sequences, our method gets the best ACA and shows absolute superiority in BWT values. These results demonstrate that the proposed method introduces a supervisory mechanism to guide network expansion, whose growth size fully considers the intrinsic complexity of each task sequentially presented.

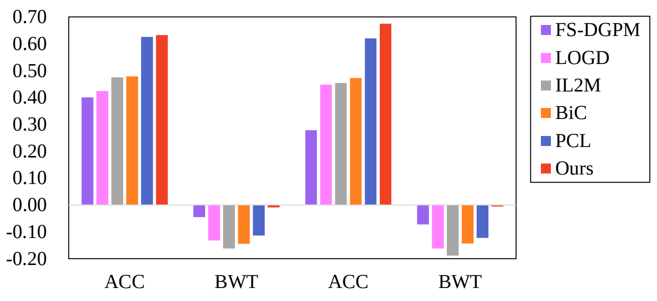

Figure 3. Performance comparison on the intra-sequence imbalanced case. *Left:* CIFAR-$\{$100(10), 100(20), 100(30), 100(40)$\}$; *Right:* CIFAR-$\{$100(2), 100(4), 100(6), ...$\}$.

D.2. Experiments on the Scalability for CIL

To show the scalability of our method, we further display the changes of model sizes during sequential training. All methods are built upon the same backbone architecture ResNet-18 ($\sim$44.6MB). Note that RPS-Net and MORE need extra exemplar buffers. We report the actually involved model size (i.e., paths) for RPS-Net. It can be observed from Figure 4 that the proposed method outperforms the competitors. That is, after incrementally learning 10 tasks, it remains the model size relatively unchanged, without relying on exemplar buffers. We believe this is promising for practical CIL under resource-limited and privacy-sensitive scenarios.

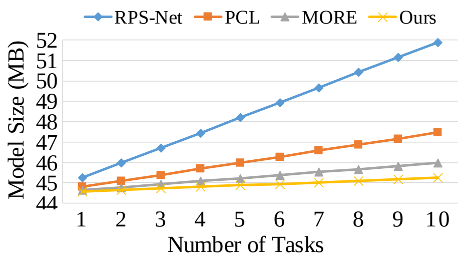

Figure 4. Growth of the network as the number of tasks increases for CIFAR-100/10.

D.3. Experiments on the Training Time and Computational Costs

As the proposed method uses a supervisory mechanism-based node recruitment step followed by a weight update step (where matrix inversion is needed), it is natural to think of the training time cost, e.g., comparing it with simpler rehearsal-based methods. Since different methods have very different requirements in computational burden, Table 8 records their per-epoch running time, demonstrating our superiority in developing a fast and easy-implementation CIL model under selected parameters ($l = 10$, $T_{max} = 50$). As our work uses a dynamic network architecture, we also provide the computational costs measured by floating point operations per second (FLOPS). The calculation of FLOPs is affected by the size of the network and input. For a fair comparison, we follow the exact settings in AANet (Liu et al., 2021), with ResNet-32 for 32 × 32 CIFAR-100 and ResNet-18 for 224 × 224 ImageNet-Subset. The results yielded from ours and AANets are 70.12 M (1.82 G) and 140.00 M (3.64 G) on CIFAR-100 (ImageNet-Subset), respectively. This demonstrates the effectiveness and efficiency of our method in constructing dynamic network architecture for CIL.

Table 8. Comparison results with rehearsal-based methods on running time per epoch.

METHOD	GEM	LOGD	RPS-NET	IL2M	FS-DGPM	OURS
TIME (S)	48	333	56	50	63	12

D.4. Experiments on Single CIL and Online CIL

In addition to the standard CIL scenario, here we consider another two variants. One is single-class incremental learning that learns one class at a time. This is the most common case in practice because once a new class is encountered, we want to learn it immediately rather than wait for a few new classes to occur and learn them together. Following the setting in PCL (Hu et al., 2021), Table 9 reports the comparison results for the Single CIL. In this scenario, the results except ours are drawn from that of PCL. It's worth mentioning that PCL, namely per-class continual learning, particularly excels at class-incremental learning one-by-one. For the CIFAR-100/100 (100 tasks), PCL yields 63.61% and the proposed method is superior to it by 1.87%.

Another is Online CIL, in which a model learns new classes continually and data can only be observed once. In this scenario, the results except ours are taken from their original papers and the four competitors need the exemplar buffer. It is observed from Table 9 that our method outperforms ASER (Shim et al., 2021), PRC (Lin et al., 2023), OnPro (Wei et al., 2023), GSA (Guo et al., 2023) on CIFAR-100/10 with fewer memory (buffer) usage. This demonstrates the effectiveness of the proposed method in the Online CIL scenario.

D.5. Comparison Results When Using More Additional Parameters

In the main text, Table 3 adopts a ViT-B/16 transformer model pre-trained on ImageNet-21K and then incrementally learns tasks from the well-established 200-class Split Imagenet-R. In this setting, our method outperforms the selected baselines by

Table 9. Comparison results on the CIFAR-100/100 for Single CIL and CIFAR-100/10 for Online CIL.

Single CIL		Online CIL	
Method	ACA(%) ↑	Method	ACA(%) ↑
EWC	2.93	ASER	14.0
RPS-Net	4.13	PCR	25.6
OWM	63.26	OnPro	30.4
PCL	63.61	GSA	31.4
Ours	65.48	Ours	32.5

Table 10. Comparison results when more additional parameters are built upon a pre-trained model.

Method	ACA(%) ↑	Additional No. Params ($\times 10^6$) ↓
SLAC	77.00	$\sim$123.7
RanPAC	77.90	$\sim$12.5
Ours*	76.32	$\sim$3.90
Ours**	78.91	$\sim$127.5
Upper bound	79.13	-

only incurring 0.6 ($\times 10^6$) additional number of parameters (Additional No. Params) except for the pre-trained ViT. Now we further compare our method with competitors using more Additional No. Params. When more additional parameters are built upon a pre-trained model, strong baselines like SLCA (Zhang et al., 2023) and RanPAC (McDonnell et al., 2023) show better performance than that of all methods in Table 3. In Table 10, when our method enlarges its model size (Additional No. Params is about 3.9×10^6), denoted by Ours*, its ACA is very approaching SLCA and RanPAC. Since our method belongs to the theoretical and embryonic approach, highlighting a fair comparison by considering both accuracy and memory usage, the result is still promising. Furthermore, as SLAC can be naturally plug-and-play with other CIL approaches, we make this combination with Ours*, denoted by Ours**. We can achieve extra performance improvement, and the results surpass SLCA and RanPAC by 1.91% and 1.01%, surprisingly getting closer to the upper bound.

D.6. Further Comparison on ImageNet-100 and ImageNet-1K Using the Metric AIA

Before concluding our empirical evaluation, Table 11 provides a comparison between the proposed method and some top-performing methods on ImageNet-100 and ImageNet-1K, measured by AIA. We first test our methods on ImageNet-100. Following the benchmark protocol used in PODNet (Douillard et al., 2020), AANets (Liu et al., 2021), DER(Yan et al., 2021), FOSTER (Wang et al., 2022a), and ACIL (Zhuang et al., 2022), we start from a model trained on 50 base classes (B50), and the remaining 50 classes are divided into splits of 10 steps. Note that we directly take the reported results on ImageNet-100 B50 from their original papers to report the best performance. It can be observed that our method is superior to the strongest baseline by 0.52%. Similarly, we test our methods on ImageNet-1K. For our method, we build it upon the available backbone provided by ACIL where a ResNet-18 was well-trained based on half of the ImageNet-1K datasets. This ImageNet-1K B500 setting is also used in PODNet, AANets, and ACIL but not in DER and FOSTER. For this, we mark DER and FOSTER with B0, i.e., training on ImageNet-1K from scratch. Although training from scratch seems more attractive, they rely on storing 20 000 old task exemplars that our method does not. It can be observed that our method outperforms the strongest baseline by 0.82%.

Table 11. Comparison results using the metric AIA(%). * denotes additional exemplar buffer is required during CIL.

Method	ImageNet-100	ImageNet-1K
PODNet*	74.33	64.13
AANets*	75.58	64.85
DER*	77.73	66.73 (B0)
FOSTER*	77.54	68.34 (B0)
ACIL	74.76	64.84
Ours	78.25	69.16