CONTINUAL LEARNING VIA ADAPTIVE NEURON SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning (CL) aims at learning a sequence of tasks without losing previously acquired knowledge. Early efforts have achieved promising results in overcoming the *catastrophic forgetting* problem. As a consequence, contemporary studies turn to investigate whether learning a sequence of tasks can be facilitated from the perspective of *knowledge consolidation*. However, existing solutions either still confront severe forgetting issues or share narrow knowledge between the new and previous tasks. This paper presents a novel Continual Learning solution with Adaptive Neuron Selection (CLANS), which treats the used neurons in earlier tasks as a knowledge pool and makes it scalable via reinforcement learning with a small margin. Subsequently, the adaptive neuron selection enables knowledge consolidation for both old and new tasks in addition to overcoming the CF problem. The experimental results conducted on four datasets widely used in CL evaluations demonstrate that CLANS outperforms the state-of-the-art baselines.

1 INTRODUCTION

Continual learning (CL), as one of the human-like lifelong learning paradigms, has received enormous attention in Artificial Intelligence (AI) community due to its capability to incrementally learn a sequence of tasks in a (deep) neural network and to keep accumulating knowledge throughout its lifetime Lange et al. (2022). In contrast to traditional online deep learning paradigms limited to learning multiple tasks in a specific domain and/or using static samples, CL confronts the problem of *catastrophic forgetting* (CF) due to the data distribution shifts McCloskey & Cohen (1989); French (1999) like, for example, arrival of brand new datasets with different input and label space. Existing CL solutions can be broadly categorized in: (i) regularization-based Kirkpatrick et al. (2017); Jung et al. (2020); Liu & Liu (2021); (ii) experience replay-based Shin et al. (2017); Korycki & Krawczyk (2021); Yin et al. (2021); and (iii) network adaption-based Rusu et al. (2016); Niu et al. (2021); Liu et al. (2021); Qin et al. (2021). In practice, these approaches resort to resolving the *stability-plasticity* concern Mermillod et al. (2013) by either manipulating the data distribution or altering the neural architecture, including weight regularization. As a result, studies that concentrate on the CF have achieved remarkable results Parisi et al. (2019); Mirzadeh et al. (2022).

Inspired by human cognitive processes Hupbach et al. (2007), we argue that tackling a series of tasks in CL is similar to the dichotomy in human experience learning, in that: (1) the learning of new tasks heavily depends on the perspective of existing experiences; and (2) the skills learned earlier could be cognitively altered or enhanced by the arrival of new knowledge. That is to say, incremental knowledge acquisition could have a corrective effect on early cognition. Thus, concomitantly with CF, knowledge consolidation is also critical for CL as it enables to comprehend the potential (back and forward) knowledge transfer between old and new tasks.

Recent studies have investigated the phenomenon of knowledge acquisition behind task learning and enforce knowledge consolidation in CL to boost the learning of old and new tasks Ke et al. (2020b; 2021); Kang et al. (2022); Wang et al. (2021). For instance, Benavides-Prado et al. (2020) and Ke et al. (2021) show that forward knowledge transfer helps promote the learning of new coming tasks. Meanwhile, Ke et al. (2020a) demonstrated that knowledge sharing can be enhanced by measuring the data similarity between the new and old tasks. Network adaptations attempting to adjust the neural network have also gained attention for knowledge consolidation Rusu et al. (2016); Li et al. (2019). For instance, HAT Serra et al. (2018) and WSN Kang et al. (2022) selectively reuse the optimized parameters via a binary masking mechanism, enabling significant forward knowledge transfer as well as no CF issue. However the backward knowledge transfer is not considered. Reinforcement

learning (RL) based solutions such as RCL Xu & Zhu (2018) make the task networks expandable, primarily learning new knowledge only on added new parameters, while retaining used parameters as unchangeable. However, they confront severe network redundancy as the built models need to reuse all of the old parameters for new task training. BNS as a complex system Qin et al. (2021) also leverages RL to dynamically provide multiple actions for the expandable task networks, allowing for backward transfer. However, the RL's actions significantly disturb the current (even future) task performance during backward transfer (and network redundancy is not considered).

In this study we investigate the possible knowledge transfer problem in CL based on the network adaptation paradigm, in-concert with the CF concern. The rationale is that most existing regularization-based or experience-replay-based methods typically confront a non-negligible and severe forgetting problem as the number of tasks increases. As a basic unit in a neural network, the neuron can be regarded as an information hub that is capable of sending messages via synapses (weights) to another neuron. To make knowledge transferable and reduce the network redundancy, we propose a novel **C**ontinual **L**earning solution via **A**daptive **N**euron **S**election (**CLANS**). CLANS regards the used neurons as a shareable knowledge pool that can be dynamically expanded by an RL, and we call such RL a *Network Expander*. Specifically, RL first provides how many new neurons are available for the arriving task, where the state of RL is determined by the similarities between the current task and the previously learned tasks. We argue that using RL to provide a neuron-level selection will result in a huge search space and cost more unnecessary resources. Thus, an expandable gating mechanism, namely *Network Selection*, undertakes the role of determining which neurons (both new and old) are useful for current task. In particular, motivated by HAT Serra et al. (2018) and WSN Kang et al. (2022), we make each layer in the task network associated with a layer embedding – noting that there is a significant difference: ours is an expandable embedding, which enables neuron selection dynamically. When the architecture of the current task is determined and trained well, our knowledge pool will be correspondingly enriched. To enable knowledge transfer to old tasks, we present a neuron selection refresh method that seeks a better architecture for each previous task from the augmented knowledge pool. As corresponding weights of each neuron in the knowledge pool are unchangeable, this will not affect the performance of the current task. Thus, in comparison with existing works such as HAT, RCL, and WSN, our method not only takes into account the complexity of the task network but also makes both forward and backward knowledge transferable. The experiments conducted on four datasets demonstrate promising gains over several baselines, as well as task network complexity. Our main findings are:

- The collaboration between Network Expander and Network Selection in CLANS can make the target network more compact, yielding efficient neuron usage with the profitable task performance;
- CLANS operates the Network Expander in REINFORCE, requiring only a few new neurons, but offering a higher possibility of recruiting more unknown knowledge from the current task;
- Compared to existing solutions that make network architecture of each task fixed, the refreshable neuron selection in CLANS enables knowledge transfer from new to old tasks;
- In our experiments, the task networks determined by CLANS, whether fully connected or convolutional, achieve superior average accuracy performance as well as (back and forward) knowledge transfers across multiple tasks, e.g., up to 1.05% and 33.33% improvements compared to the best baseline on CIFAR-100 regarding Accuracy (ACC) and Forward Transfer (Trans).

2 Related Work

We now review the related work from two basic categories and position CLANS in that context.
(1) Catastrophic Forgetting (CF): Most of the previous endeavors focus on overcoming the CF and the existing studies can be divided into three main groups: (i) *Regularization-based learning* aims at penalizing or limiting the weight optimization for tackling the new coming task, primarily seeking to make the important weights for earlier tasks not altered too much, and researchers have striven to develop efficient methods for important weight measurements during sequential task learning Kirkpatrick et al. (2017); Zenke et al. (2017); Lee et al. (2017); Ahn et al. (2019); Zhang et al. (2020); Jung et al. (2020). For instance, Elastic Weight Consolidation (EWC) Kirkpatrick et al. (2017) uses an additional regularization term based on the diagonal of the Fisher information matrix in the task-oriented objective to alleviate the forgetting problem of old knowledge and Zenke et al. (2017) introduces the synaptic intelligence (SI) to measure weight importance. Other efforts

attempted using masks to freeze some of the used weights Mallya & Lazebnik (2018); Mallya et al. (2018). (ii) *Experience Replay* aims to build a sample buffer or knowledge base (KB) to preserve useful/important samples from the old task and co-train them with the future task Shin et al. (2017); Isele & Cosgun (2018); Rolnick et al. (2019); Wang et al. (2022). (iii) *Network Adaption (or Parameter Isolation* attempts to allocate new neurons/weights on an earlier neural network for training on new tasks, whereby each task will have its individual and shared parameters Rusu et al. (2016); Yoon et al. (2018); Xu & Zhu (2018); Serra et al. (2018); Banayeeanzade et al. (2021). For instance, Xu & Zhu (2018) and Qin et al. (2021) leverage reinforcement learning to allocate the new neurons for each new task. Serra et al. (2018) and Kang et al. (2022) develop the masking methods to select useful neurons or connections for new coming task training while freezing the neurons or connections used in the earlier tasks. In contrast, CLANS not only selects useful neurons and perform small network expansions, but also dynamically reconfigure the task network architecture of earlier tasks. *(2) Knowledge Acquisition*: Recently, researchers and practitioners have argued that the essence of CL is to achieve knowledge accumulation or consolidation that further enables helping the learning of future unfamiliar tasks. Thus, several works concentrated on transferring the learned knowledge to facilitate the learning in the new domains or new tasks, e.g., Bayes model Chen et al. (2015); Wang et al. (2020) and regression method Ruvolo & Eaton (2013). Conversely, some studies consider leveraging the future tasks to help enhance the performance of old tasks Wang et al. (2019). While striving to select useful knowledge from the past (future) to boost the learning of future (past) tasks, each task is assigned an independent learning model or knowledge base, which naturally leads to the capacity problem. In deep learning scenarios, most existing neural networks have similar limits (i.e., extremely forgiving the capacity issue) when tackling knowledge transfer concerns Rusu et al. (2016); Silver et al. (2013); Ke et al. (2020b). Instead, CLANS is flexible for knowledge consolidation, enabling either back or forward knowledge transfer, and the adaptive neuron selection in CLANS provides us with a more compact network for each task learning.

3 METHODOLOGY

3.1 PROBLEM FORMULATION AND BASIC WORKFLOW

Given a task T_t ($t \in \{1, 2, \cdots, T\}$), it has a training set $\mathcal{D}_t^{train} = \{(x_t^i, y_t^i)\}_{i=1}^{O_t}$, where t is the task id, O_t is the scale of training set, any x_t^i refers to the source instance (input), and y_t^i denotes the target (class label). Similarly, task t also has a validation set $\mathcal{D}_t^{valid}$ and a test set $\mathcal{D}_t^{test}$. Formally, we set our CL context as follows: assume there are $t-1$ tasks that have been incrementally/progressively learned by the task network, until now. When a new task T_t is coming, the goal of CLANS is to dynamically allocate new neurons to the knowledge pool and adaptively select new and old neurons from the knowledge pool for task T_t training, where the corresponding parameters of old neurons cannot be optimized. After finishing the training of task T_t, CLANS will try to employ the updated knowledge pool to adjust each previous task's architecture for knowledge backward transfer.

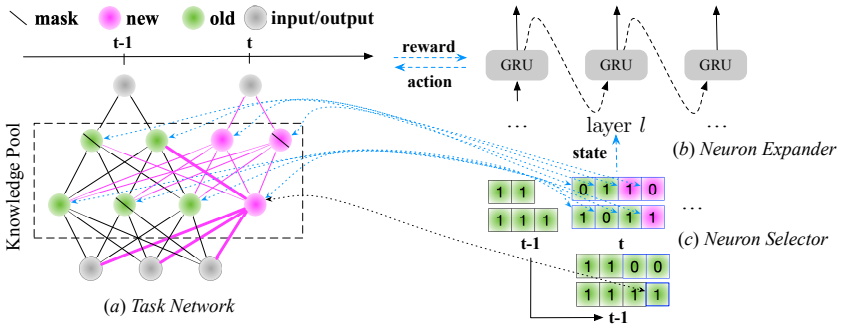


Figure 1: The main components in CLANS.

CLANS is a reinforcement learning-based method that mainly has three components, i.e., *Task Network*, *Neuron Expander*, and *Neuron Selector*. Especially, we regard the hidden layers in Task Network as the Knowledge Pool ($\mathcal{KP}$) which aims to preserves the neurons (as well as their corresponding parameters) that have been used in all of the previous tasks. As shown in Fig. 1, each

green circle denotes the neuron that has been used in the previous tasks, and each red neuron denotes the newly added neurons that can be used for task T_t. Note that each neuron is associated with a selection score derived from the layer embedding (the blocks in Neuron Selector). During the training of current task T_t, Neuron Selector will use the gate operation to select part of neurons for task training. That is to say, the circle with '\' will not be used.

When a new task T_t comes, we attempt to search for the best network architecture for task T_t in the reinforcement learning context, we first compute the similarity score between the current task and each of the previous tasks based on the replay buffer $\mathcal{B}_{t-1}$, yielding a similarity set. Correspondingly, we treat this similarity set as the initial *state* of the Neuron Expander, and use it to obtain an action sequence that will determine how many new neurons in each layer are available for the current task T_t. After that, we can allocate new neurons into $\mathcal{KP}$ and use Neuron Selector to pick the useful neurons for current task training while each layer embedding in current task network will also be updated. Next, we can calculate a reward that considers the model accuracy and complexity, and use it to update the Neuron Expander. To make the *state* dynamically reflect the architecture of built task network, we turn to use the a set of layer embedding similarities instead of data similarities as the next state of Neuron Expander, where layer embedding is obtained from the Neuron Selector. We note that such interactions repeat until a good architecture is found. Then, when the architecture of the current task T_t is determined and well trained, the augmented $\mathcal{KP}$ containing some newly added neurons is used to adjust/refresh each previous task's architecture by optimizing their layer embedding again. As shown in the lower right corner of Fig. 1, we first expand the layer embeddings of task $t-1$ to make newly added neurons available for task T_t. Next, we employ the replay buffer to search out a better architecture for task $t-1$. For other earlier tasks, we do this in a sequence. Note that CLANS can only use the training samples from the current task T_t while the training samples in the previous tasks are no longer available. To measure the task similarity and evaluate the gains of knowledge transfer, we will add each task's validation set to an experience replay set, detailed next.

3.2 Task Network and Neuron Selector

Let $\mathcal{F}_t$ denote a task network that aims to model the given task T_t. Basically, we also denote $\mathcal{F}_t = (\mathcal{N}_t, \mathcal{W}_t)$ where $\mathcal{N}_t$ are the neurons used for training task T_t, and $\mathcal{W}_t$ are the corresponding weights. When a new task T_t $(t > 1)$ is coming, we have learned $t-1$ tasks well, where the neurons of every task network are determined, i.e., $\mathcal{N}_i$ $(i < t)$, and their corresponding weights are also optimized, i.e., $\overline{\mathcal{W}}_i$. In CLANS, the architecture of task network $\mathcal{F}_t$ is determined by our Neural Expander and Neural Selector. We correspondingly need to optimize the network architecture $\mathcal{F}_t = (\mathcal{N}_t, \mathcal{W}_t)$ for task T_t. Specifically, there exist two types of neurons in $\mathcal{N}_t$: $\mathcal{N}_t^{old}$ are selected from the used neurons from earlier tasks while $\mathcal{N}_t^{new} = \mathcal{N}_t \setminus \mathcal{N}_t^{old}$ are newly added neurons. Similarly, the corresponding weights can also be divided into two parts, i.e., optimized weights $\overline{\mathcal{W}}_{t-1}^{old}$ and the new weights W_t^{new}. Therefore, we can obtain the objective of the current task as:

$$\overline{\mathcal{W}}_t = \arg\min_{\mathcal{W}} \mathbb{E}_{(x,y)\sim\mathcal{D}_t^{train}} \left[\mathbb{L}\left(\mathcal{F}_t\left(x; \mathcal{W}_t|_{\mathcal{W}_t^{old}=\overline{\mathcal{W}}_{t-1}^{old}} \right), y \right) \right], \tag{1}$$

where $\mathbb{L}$ is the loss function such as cross-entropy. $\overline{\mathcal{W}}_t$ are optimal parameters after convergence.

Inspired by HAT Serra et al. (2018), our Neuron Selector uses the layer-wise gate mechanism conditioned on layer embedding to decide which neurons will be used for task training. The gate operation can be defined as the follows:

$$\boldsymbol{g}_t^l = \sigma(\gamma \cdot \boldsymbol{e}_t^l), \tag{2}$$

where t is the task id, σ is a gate function (e.g., sigmoid), γ is the hyper-parameter, and $\boldsymbol{e}_t^l$ is the l-th layer embedding. Note that we define a cumulative gate vector $\boldsymbol{m}_{\leq t}^l$ to preserve the information learned in previous tasks. Specifically, the cumulative gate vector can be recursively obtained by:

$$\boldsymbol{m}_{\leq t}^l = \max(\boldsymbol{m}_{\leq t-1}^l, \overline{\boldsymbol{g}}_t^l), \tag{3}$$

where $\overline{\boldsymbol{g}}_t^l$ is the optimal results after model convergence. Herein, we set $\boldsymbol{m}_{\leq 0}^l = \mathbf{0}$.

3.3 Neuron Expander and Training with Knowledge Transfer

We use Neuron Expander to decide how many neurons should be added to each layer after the task t arrives. Suppose the Task Network has L hidden layers, Neuron Expander should specify

the number of neurons to add in the range between 0 and n_l for each layer l. As mentioned in the previous studies Xu & Zhu (2018); Qin et al. (2021), enumerating possible number of newly added neurons for each layer is usually NP-hard, i.e., $\mathcal{O}(\prod_1^L n_l)$. To this end, we only employ the reinforcement learning and regard such a series of actions as a fixed-length string that specifies how many neurons should be added in each layer. As there is a recursive correlation between adjacent layers in the task network, Neuron Expander should intrinsically operate in a recurrent manner. Hence, we utilize popular recurrent neural networks (e.g., LSTM Hochreiter & Schmidhuber (1997) or GRU Cho et al. (2014)) as the kernel of Neuron Expander. In our implementation, we operate the GRU in an autoregressive manner for efficiency. We follow the standard reinforcement learning setup to illustrate the technical details:

– State. It is to perceive the work environment of reinforcement learning. When task T_t arrives, we strive to measure the task similarity between the current task and the previous tasks. In contrast to prior study Xu & Zhu (2018), we treat the scores of data distribution similarities as the initial states in RL instead of random initialization. Let $\boldsymbol{s}_t^0 = [s_{t,0}^1, s_{t,0}^2, \cdots, s_{t,0}^{t-1}]$ refer to the initial states, we use *Jensen-Shannon Divergence* (JSD) to measure the similarity between the current task T_t and an earlier task T_i ($i = 1, 2, \cdots, t-1$). For instance, any state $s_{t,0}^i$ can be computed by:

$$JSD(P_t(\mathcal{D}_t^{valid}) \| P_i(\mathcal{D}_i^{valid})) = \frac{1}{2} KL(P_t(\mathcal{D}_t^{valid}) \| \bar{P}_{ti}) + \frac{1}{2} KL(P_i(\mathcal{D}_i^{valid}) \| \bar{P}_{ti}), \tag{4}$$

where $\bar{P}_{ti} = \frac{P_t(\mathcal{D}_t^{valid}) + P_i((\mathcal{D}_i^{valid}))}{2}$. Note that we compute the task similarity between task T_t and T_i by using the validation set of task T_t (i.e., $\mathcal{D}_t^{valid}$) and the task T_i's replay set $\mathcal{D}_i^{valid}$ preserved in replay buffer $\mathcal{B}_{t-1}$. We can clearly find that $\boldsymbol{s}_t^0$ do not change with the actions in RL, resulting the training gap between *state* and *action*. Fortunately, our Neuron Selector provides us with an opportunity that we can further use the selected neuron results to measure the task similarities. Without loss of generality, we can measure the similarity between the current task T_t after $j-1$-th iteration of RL and a previous task T_i by:

$$k_{t,j}^i = \textit{cosine}([\boldsymbol{g}_{t,j}^1, \boldsymbol{g}_{t,j}^2, \cdots, \boldsymbol{g}_{t,j}^L], [\boldsymbol{g}_i^1, \boldsymbol{g}_i^2, \cdots, \boldsymbol{g}_i^L]), \tag{5}$$

where $k_{t,j}^i$ denotes the similarity score between the task T_t and the task T_i after $j-1$-th iteration, *cosine* is the cosine distance. As such, we obtain the current state as: $\boldsymbol{s}_t^j = [k_{t,j}^1, k_{t,j}^2, \cdots, k_{t,j}^{t-1}]$, which will be used in the j-th iteration of RL.

– Action. When Neuron Expander receives the current state $\boldsymbol{s}_t^j$, we use it as the first hidden state of the GRU and recursively generate a series of actions $\boldsymbol{a}_t^j = \{a_{t,j}^l\}_{l=1}^L$to determine how many neurons will be added to each layer of the task network. The procedure is summarized as follows:

$$\boldsymbol{c}_{t,j}^l = \text{GRU}_\psi(\boldsymbol{c}_{t,j}^{l-1}, a_{t,j}^{l-1}), a_{t,j}^l = \arg\max \Phi(\boldsymbol{c}_{t,j}^l \boldsymbol{W}_a + \boldsymbol{b}_a), \tag{6}$$

where $\boldsymbol{c}_{t,j}^0 = \boldsymbol{s}_t^j$ and Φ denotes the *Softmax* function. Besides, ψ, $\boldsymbol{W}_a \in \mathbb{R}^{d \times n_l}$ and $\boldsymbol{b}_a \in \mathbb{R}^{n_l}$ are trainable parameters. The above procedure is circulated until we have received the actions for all L expandable layers. Formally, we denote the policy function by $\pi(\boldsymbol{a}_t^j | \boldsymbol{s}_t^j; \theta)$ as:

$$\pi(\boldsymbol{a}^{t,j} | \boldsymbol{s}_t^j; \theta) = P_a(a_{t,j}^1 | \boldsymbol{s}_t^j; \theta) \prod_{l=2}^{L} P_a(a_{t,j}^l | a_{t,j}^{l-1}; \theta), \tag{7}$$

where θ denotes a set of learnable parameters in the Neuron Expander.

– Reward. In our context, a reward function not only needs to reflect the task accuracy but also enables architecture complexity evaluation. To this end, we evaluate RL performance in each iteration using accuracy improvements and network increments, which can be denoted as:

$$R_t^j = \frac{acc_t^j - acc_t^{j-1}}{C(\mathcal{N}_t^j) - C(\mathcal{N}_t^{j-1})}, \tag{8}$$

where acc_t^j is validation accuracy of j-th iteration on the validation set $\mathcal{D}_t^{valid}$ of task T_t and $C(\mathcal{N}_t^j)$ is used to calculate the number of used neurons.

We now proceed to illustrate how CLANS transfers knowledge to current and previous tasks, with a note that the algorithmic pipeline is detailed in Appendix 1.

REINFORCE for Current Task. When task T_t arrives, we use Neuron Expander in RL to decide the number of new neurons that can be added to each layer l, e.g., $a^l_{t,j}$ (for simplicity, we omit the iteration index of RL). We temporarily add these neurons to our knowledge pool $\mathcal{KP}$ where each layer l will contain neurons $\mathcal{N}^l_{\leq t} = \{\mathcal{N}^l_{<t-1}, \mathcal{N}_{a^l_t}\}$. Before training task T_t, we should extend the dimension of the cumulative gate vector $\boldsymbol{m}^l_{\leq t-1}$ obtained from the earlier task T_{t-1}. Specifically, we set $\boldsymbol{m}^l_{\leq t} = [\boldsymbol{m}^l_{\leq t-1}, \mathbf{0}(a^l_t)]$, indicating that the extend neurons can be used for the current task. Next, we prepare task T_t's layer embeddings where formulating each layer l by $\boldsymbol{e}^l_t = [e^1_t, e^2_t, \cdots, e^{C(\mathcal{N}^l_{\leq t})}_t]$. Then we use Eq.(2) to make neuron selection for task T_t. Assume that $\boldsymbol{h}^l_t$ refers to the outputs of neurons in layer l of task T_t, the forward propagation in each task training epoch can be summarized as follows:

$$\textbf{forward: } \boldsymbol{h}^l_t = \boldsymbol{h}^l_t \odot (\boldsymbol{g}^l_t \odot \neg \boldsymbol{m}^l_{\leq t}), \tag{9}$$

where $\neg$ is *Negate* operation. Meanwhile, the backward propagation can be obtained by:

$$\textbf{backward: } \mathcal{W}_{l,u,v} = \mathcal{W}_{l,u,v} - \left[\max(g^{l,u}_t \odot \neg m^{l,u}_{\leq t}, g^{l-1,v}_t \odot \neg m^{l-1,v}_{\leq t})\right] \odot \frac{\partial \mathbb{L}}{\partial \mathcal{W}_{l,u,v}}, \tag{10}$$

where neuron indices u and v correspond to layer l and layer $l-1$, respectively. After task training convergence, we can obtain the final selection decision $\boldsymbol{g^*}^l_t$ for current task T_t by:

$$\boldsymbol{g^*}^l_t = \sigma(\gamma \cdot \boldsymbol{e^*}^l_t). \tag{11}$$

Notably, each iteration of RL will obtain an immediate selection decision, and we always use $*$ to refer to the decision result in each iteration for convenience. When the RL algorithm terminates, we regard the decision result corresponding to the maximum reward as the optimal choice, e.g., $\overline{\boldsymbol{g}}^l_t$. And we obtain the final cumulative gate vector $\mathbf{m}^l_{\leq t}$ and the optimal neuron number $\bar{a}^l_t$ for each layer l.

Selection Refresh for Previous Tasks. As we mentioned above, our knowledge pool $\mathcal{KP}$ is enriched by learning from new tasks T_t. Hence, we conjecture whether we can use the updated pool to promote the performance of old tasks. That is to say, we resort to using $\mathcal{KP}$ to promote previous tasks. When we have determined the architecture of task T_t, we extend each previous task's layer embedding for the purpose of receiving new knowledge transferred from the updated knowledge pool. For each layer l in an earlier task T_i ($i = 1, 2, \cdots, t-1$), its layer embedding can be set by:

$$\boldsymbol{e}^l_i = \overline{\boldsymbol{e}}^l_i || [e_1, e_2, \cdots, e_{\bar{a}^l_t}]. \tag{12}$$

To determine the selection gap between task T_t and task T_i, we use the following operation to make knowledge transfer happen on new neurons that are finally selected by task T_t:

$$\boldsymbol{g'}^l_i = \sigma(\gamma \cdot \boldsymbol{e}^l_i \odot \boldsymbol{o}^l_i), \boldsymbol{o}^l_i = \min(\neg(\boldsymbol{g}^l_i || \underbrace{[0, 0, \cdots, 0]}_{\bar{a}^l_t}), \boldsymbol{g}^l_t), \tag{13}$$

where $\boldsymbol{g'}^l_i$ denotes the new select decision for task T_i and each 0 indicates that its corresponding neuron can be used now. To protect the model performance of the current task T_t, no paramter updating process (i.e., forward propagation) is allowed in the knowledge pool, and we let each earlier task's classifier to be trainable again to fine-tune its performance. More importantly, we withdraw knowledge transfer if an earlier task performs poorly on task-related samples in the replay buffer, as we do not allow the knowledge transfer to significantly disturb its earlier determined structure.

4 EXPERIMENTS

Datasets and Baselines. We conduct the experiments on four datasets that are widely used in CL evaluation. They are MNIST Permutation (PMNIST) Kirkpatrick et al. (2017), Rotated MNIST (RMNIST) Xu & Zhu (2018), Incremental CIFAR-100 Rebuffi et al. (2017), and TinyImageNet Kang et al. (2022). PMNIST contains 10 variants of the MNIST LeCun (1998), where each task is transformed by a different and fixed permutation of pixels. RMNIST also has ten versions of MNIST, where each task is rotated by a different angle between 0 to 360 degrees. For Incremental CIFAR-100, we split orignal CIFAR-100 Krizhevsky et al. (2009) into 10 tasks and each

contains 10 different classes. TinyImageNet is a variant of the ImageNet Kirzhevsky et al. (2012). For consistency in our experiments, we generate ten 20-way classification tasks. We meticulously select powerful representative baselines for comparison: **SGD** Goodfellow et al. (2013) is a naive method that uses a feature extractor without any parameter update and a learnable classifier to tackle a series of tasks incrementally. **EWC** Kirkpatrick et al. (2017) is a simple and effective standard baseline that uses Fisherman information to alleviate the CF. **IMM** Lee et al. (2017) is motivated by transfer learning to penalize parameter modifications. More specifically, we report two IMM variants in our experiments, i.e., mean-IMM and mode-IMM. **PGN** Rusu et al. (2016) attempts to expand the task network by adding a fixed number of neurons. **DEN** Yoon et al. (2018) dynamically decides the number of new neurons by performing selective retraining and network split. **RCL** Xu & Zhu (2018) is a reinforcement learning method that aims to control the scale of newly added neurons while making the weights w.r.t. used neurons unchangeable. **HAT** Serra et al. (2018) employs attention-based gate operation to prevent the optimization of old neurons. **SupSup** Wortsman et al. (2020) enables sequentially learning tasks without the CF problem by finding supermasks (sub-networks). **WSN** Kang et al. (2022) attempts to sequentially learn the model parameters and selects an optimal sub-network for each task.

Implementation Setup and Evaluation Metrics. To follow the manner of task-incremental CL Xu & Zhu (2018); Serra et al. (2018); Kang et al. (2022), all methods use multi-head configuration in all experiments. For two MNIST variants, we follow the setup in Serra et al. (2018) and use a two-layer MLP and a multi-head classifier, starting with 2000-2000-10 neurons for the first task. For the remaining datasets, we follow previous works Serra et al. (2018); Kang et al. (2022) and use the modified AlexNet Kirzhevsky et al. (2012) which contains three convolutional layers and three fully-connected layers. Notably, more implementation details are shown in Appendix B.2.3, including datasets, metric protocols, architectures, and experimental settings. And the source code can be found in Supplementary Materials. We follow recent continual learning baselines Qin et al. (2021); Kang et al. (2022) and report three metrics including Average Accuracy (**ACC**), Backward Transfer (**BWT**), and Forward Transfer (**Trans**). **ACC** is a common CL metric that reports the average accuracy of all tasks validated on their respective test sets after all tasks have been trained. **BWT** is a forgetting measurement of how the model performance of the old task changes after training for the new task. **Trans** is a measurement of the forward transfer ability compared to the independent training, indicating how useful knowledge learned from the old tasks to new task learning.

4.1 Experimental Results

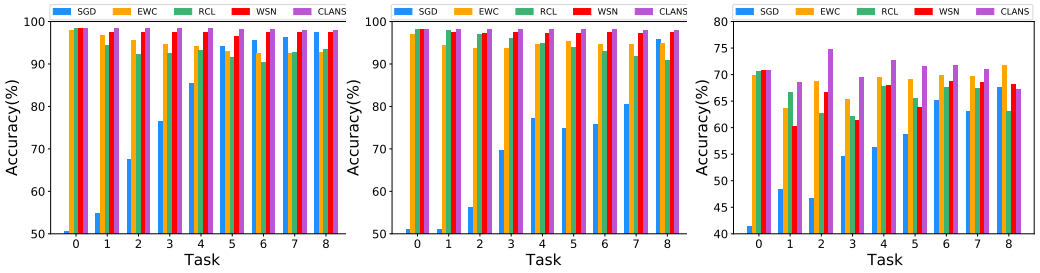
Task Performance. Table 1 reports the averaged results over 5 runs. To consider the influence of task mixture, we shuffle the tasks with 5 different seeds, resulting in 5 lists with different task orders. Specifically, we have the following observations. Among the baselines, we are surprised to find that not all CL methods significantly outperformed SGD. Even mean-IMM performs slightly worse on PMNIST than SGD. We consider the plausible reason is that the weight penalty via multiple transfer learning techniques may heavily impede the knowledge transfer between old and new tasks. CL methods with neuron expandability, e.g., PGN and RCL, also do not perform as well as EWC, which indicates that only iterative expanding the network without network pruning or neuron selection could bring the over-parameterized risk Han et al. (2015); Frankle & Carbin (2019). HAT, SupSup, and WSN follow the merit of network pruning and primarily seek to produce a sub-network by partially selecting some neurons (weights), resulting in higher accuracy performance. However, they can only make forward knowledge transfer, yielding a narrow knowledge consolidation. In contrast, CLANS makes adaptive neuron selection from an expandable knowledge pool, enabling knowledge consolidation from both old and new tasks and achieving 0.11%, 0.04%, 1.04%, and 0.40% improvements over the best baselines on four datasets, respectively. In addition, the deviation results are reported in Appendix C.3.

Knowledge Transfer. The experimental results on BWT demonstrate that SGD confronts severe knowledge forgetting issue as it does not consider the CF problem in sequential task training. Other traditional CL approaches such as EWC and IMM also have obvious forgetting issue. The reason is that recursive parameter modification for new task learning can significantly affect the model fitting ability to old task. Compared to the forget-free (BWT=0) methods, CLANS can even positively promote the previous task performance on PMNIST, CIFAR-100, and TinyImageNet (BWT$>$0). the

Table 1: Performance comparison of the proposed method and baselines on four datasets.

Model	PMNIST			RMNIST			CIFAR-100			TinyImageNet		
	ACC(%)	BWT(%)	Trans(%)	ACC(%)	BWT(%)	Trans(%)	ACC(%)	BWT(%)	Trans(%)	ACC(%)	BWT(%)	Trans(%)
SGD	81.37	-24.52	-17.06	72.83	-25.32	-25.08	57.86	-20.32	-15.77	13.60	-25.32	-25.14
EWC	94.20	-0.32	-4.23	94.86	-0.73	-3.05	68.12	-1.74	-5.51	35.24	-1.34	-2.50
mean-IMM	80.10	-1.13	-18.33	88.81	-0.96	-9.10	60.08	-0.72	-13.55	26.57	-2.89	-11.17
mode-IMM	93.13	-4.17	-5.30	89.48	-7.40	-8.43	61.50	-15.58	-12.13	27.05	-14.40	-10.69
PGN	91.89	0.00	-6.54	90.01	0.00	-7.90	58.28	0.00	-15.35	33.56	0.00	-4.18
DEN	91.96	-0.41	-6.47	91.53	-0.52	-6.38	59.32	-1.24	-12.79	33.86	-1.30	-3.88
RCL	92.28	0.00	-6.15	93.97	0.00	-3.94	61.07	0.00	-12.56	34.12	0.00	-3.62
HAT	98.10	0.00	-0.33	97.89	0.00	-0.02	70.67	0.00	-2.96	37.92	0.00	0.18
SupSup	96.32	0.00	-2.11	97.15	0.00	-0.73	68.98	0.00	-4.65	37.28	0.00	-0.46
WSN	96.46	0.00	-1.97	97.32	0.00	-0.59	69.34	0.00	-4.29	37.41	0.00	-0.33
CLANS	**98.21**	**0.01**	**-0.22**	**97.93**	**0.00**	**0.02**	**71.41**	**0.02**	**-2.22**	**39.60**	**0.01**	**1.86**

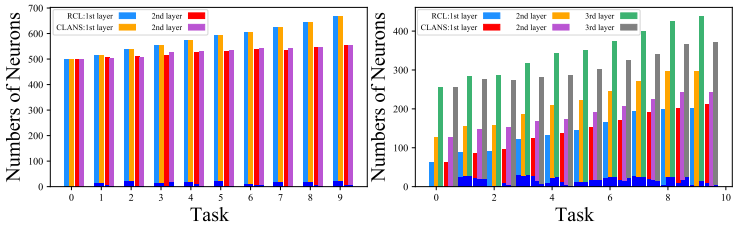
reason is that CLANS keeps the weights regarding the selected neurons from the knowledge pool unchangeable while the refreshable neuron selection on old tasks could refine the similar knowledge to enhance the old task performance. In addition, Trans is a measure that quantitatively shows whether the prior tasks can help the learning of a new task. We can observe that CLANS performs the best, which indicates that CLANS has a more substantial ability to transfer knowledge from old tasks. Moreover, the Trans>0 on either RMNIST or TinyImageNet indicates that the new task performance obtained from CLANS can even surpass its accuracy in a single-task training scenario. The reason is neuron selection with an expandable knowledge pool can distill more unknown knowledge from the new task. As a result, Fig. 2 reports each old tasks' performance after finishing the continual learning. Compared to the representative methods, we can find that CLANS significantly promotes the accuracy performance of old tasks via knowledge transfer.



(a) PMNIST (b) RMNIST (c) CIFAR-100

Figure 2: Accuracy performance for each task.

Network Capacity. Both CLANS and RCL use RL algorithms to dynamically maintain expandability for new neurons. RL in RCL is only conditioned on the current task performance and does not consider any prior knowledge. We heuristically employ the task similarity and selection similarity (i.e., layer-wise embedding) to dynamically adjust the RL environment, conditioning RL not only on the current task but also on learned tasks. Thus, we attempt to investigate the expansion preference or difference between them. Fig. 3 shows the number of reused neurons and the number of expanded neurons (the bar in dark blue). Specifically, we observe: (1) CLANS drops a significant number of old neurons, demonstrating that removing redundant neurons (as well as weights) can achieve better task performance. (2) CLANS expands fewer new neurons than RCL in each task, which indicates that our RL environment conditioned on similarity measurement is capable of governing the network expandability better while providing us with superior task accuracy. In addition, calculating the total allocated parameters after training all tasks, we find that the number of total parameters is greatly reduced, but it is slightly higher than HAT due to the network expandability.



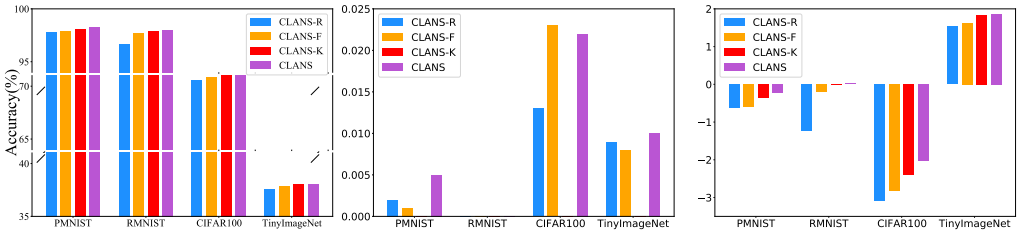
(a) PMNIST (b) CIFAR-100

Figure 3: Neuron capacity (CLANS v.s. RCL).

Table 2: Total parameters.

Dataset	EWC	RCL
P-MNIST	693.1K	1.0M
R-MNIST	693.1K	972.8K
CIFAR-100	6.7M	7.1M
TinyImageNet	6.9M	7.2M
Dataset	**HAT**	**CLANS**
P-MNIST	607.5K	652.7K
R-MNIST	335.9K	339.6K
CIFAR-100	6.6M	6.9M
TinyImageNet	6.8M	7.1M

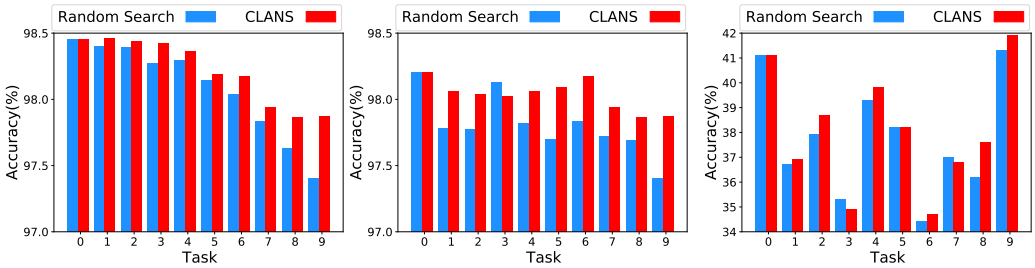
Module Effectiveness. We provide three variants of CLANS to investigate the contribution of each proposed component. The first one is **CLANS-R**, which removes the RL part and makes the task network unscalable. Although this variant has a similar gate operation for neuron selection compared to HAT, we note that this variant is significantly different from HAT as the previous tasks in ours can be trained again after finishing the new task learning. The second is **CLANS-F**, which uses a similar neuron expansion manner in PGN that enables adding a fixed number of new neurons. In our implementation, we fix the number of neuron (or filters in CNN) expansions to 30. The last variant is **CLANS-K**, which does not make selection refresh for the previous task. CLANS-K does not bring the CF problem but has no knowledge transfer for the old tasks. As Fig. 4 shows, we can observe that CLANS performs the best except for BWT on CIFAR-100. Moreover, removing any module results obvious performance drop, which verifies the effectiveness of our proposed modules.



(a) ACC (b) BWT (c) Trans

Figure 4: Module Effectiveness.

REINFORCE v.s. Random Search. In addition, we replace our RL algorithm with a random search strategy to verify the effectiveness of RL-based neuron expansion. In each interaction of RL, we randomly specify a number of added neurons in each layer. As shown in Fig. 5, we find that CLANS achieves a better performance on task accuracy.



(a) PMNIST (b) RMNIST (c) TinyImageNet

Figure 5: CLANS vs Random Search.

Efficiency. Table 3 reports the full runtime of CLANS on the four datasets. CLANS takes more time due to RL and selection refresh operations, but less than DEN.

Table 3: Statistics for full training and inference time, where 'h' represents hours.

Dataset	SGD	EWC	mean-IMM	mode-IMM	PGN	DEN	RCL	HAT	SUP	WSN	CLANS
P-MNIST	0.43h	1.06h	1.02h	1.12h	2.32h	38.24h	30.51h	0.53h	1.55h	1.92h	32.38h
R-MNIST	0.37h	0.94h	0.98h	1.04h	2.12h	20.17h	14.01h	1.49h	1.32h	1.67h	17.83h
CIFAR-100	0.07h	0.11h	0.14h	0.12h	0.52h	10.08h	7.13h	0.13h	0.12h	0.15h	8.95h
TinyImageNet	0.61h	1.04h	1.13h	1.07h	2.17h	29.43h	21.43h	0.45h	2.14h	2.31h	25.72h

5 CONCLUSIONS

We presented CLANS, a novel solution for CL via adaptive neuron selection. CLANS treats the allocated neurons from earlier tasks as a knowledge pool and makes it slightly extensible via reinforcement learning. The adaptive neuron selection enables the knowledge consolidation for both old and new coming tasks. The experiments conducted on four benchmark datasets demonstrated the significant superiority of CLANS compared to ten representative baselines, including task accuracy and knowledge transfer. In the future, we will consider designing a compact hyper-network to sequentially determine the task similarity and old task re-optimization, and reduce computational and memory costs.

REFERENCES

Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. Uncertainty-based continual learning with adaptive regularization. *Advances in Neural Information Processing Systems*, 32, 2019.

Mohammadamin Banayeeanzade, Rasoul Mirzaiezadeh, Hosein Hasani, and Mahdieh Soleymani. Generative vs. discriminative: Rethinking the meta-continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Diana Benavides-Prado, Yun Sing Koh, and Patricia Riddle. Towards knowledgeable supervised lifelong learning systems. *Journal of Artificial Intelligence Research*, 68:159–224, 2020.

Zhiyuan Chen, Nianzu Ma, and Bing Liu. Lifelong learning for sentiment classification. In *ACL*, pp. 750–756, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Almut Hupbach, Rebecca Gomez, Oliver Hardt, and Lynn Nadel. Reconsolidation of episodic memories: A subtle reminder triggers integration of new information. *Learning & memory*, 14(1-2): 47–53, 2007.

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Sangwon Jung, Hongjoon Ahn, Sungmin Cha, and Taesup Moon. Continual learning with node-importance based adaptive group sparse regularization. *Advances in Neural Information Processing Systems*, 33:3647–3658, 2020.

Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D Yoo. Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pp. 10734–10750. PMLR, 2022.

Zixuan Ke, Bing Liu, and Xingchang Huang. Continual learning of a mixed sequence of similar and dissimilar tasks. *Advances in Neural Information Processing Systems*, 33:18493–18504, 2020a.

Zixuan Ke, Bing Liu, Hao Wang, and Lei Shu. Continual learning with knowledge transfer for sentiment classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 683–698. Springer, 2020b.

Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

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.

A Kirzhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Lukasz Korycki and Bartosz Krawczyk. Class-incremental experience replay for continual learning under concept drift. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3649–3658, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(07):3366–3385, 2022. ISSN 1939-3539.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. *Advances in neural information processing systems*, 30, 2017.

Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International Conference on Machine Learning*, pp. 3925–3934. PMLR, 2019.

Hao Liu and Huaping Liu. Continual learning with recursive gradient optimization. In *International Conference on Learning Representations*, 2021.

Yaoyao Liu, Bernt Schiele, and Qianru Sun. Rmm: Reinforced memory management for class-incremental learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–82, 2018.

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

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.

Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Timothy Nguyen, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Architecture matters in continual learning. *arXiv preprint arXiv:2202.00275*, 2022.

Shuaicheng Niu, Jiaxiang Wu, Guanghui Xu, Yifan Zhang, Yong Guo, Peilin Zhao, Peng Wang, and Mingkui Tan. Adaxpert: Adapting neural architecture for growing data. In *International Conference on Machine Learning*, pp. 8184–8194. PMLR, 2021.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Qi Qin, Wenpeng Hu, Han Peng, Dongyan Zhao, and Bing Liu. Bns: Building network structures dynamically for continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019.

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.

Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *International conference on machine learning*, pp. 507–515. PMLR, 2013.

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

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*, 2013.

Hao Wang, Bing Liu, Shuai Wang, Nianzu Ma, and Yan Yang. Forward and backward knowledge transfer for sentiment classification. In *Asian Conference on Machine Learning*, pp. 457–472. PMLR, 2019.

Hao Wang, Shuai Wang, Sahisnu Mazumder, Bing Liu, Yan Yang, and Tianrui Li. Bayes-enhanced lifelong attention networks for sentiment classification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 580–591, 2020.

Liyuan Wang, Mingtian Zhang, Zhongfan Jia, Qian Li, Chenglong Bao, Kaisheng Ma, Jun Zhu, and Yi Zhong. Afec: Active forgetting of negative transfer in continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Liyuan Wang, Xingxing Zhang, Kuo Yang, Longhui Yu, Chongxuan Li, Lanqing HONG, Shifeng Zhang, Zhenguo Li, Yi Zhong, and Jun Zhu. Memory replay with data compression for continual learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=a7H7OucbWaU`.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

Ju Xu and Zhanxing Zhu. Reinforced continual learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Haiyan Yin, Ping Li, et al. Mitigating forgetting in online continual learning with neuron calibration. *Advances in Neural Information Processing Systems*, 34, 2021.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Sk7KsfW0-`.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Jie Zhang, Junting Zhang, Shalini Ghosh, Dawei Li, Jingwen Zhu, Heming Zhang, and Yalin Wang. Regularize, expand and compress: Nonexpansive continual learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 854–862, 2020.

APPENDIX

A.1 ALGORITHMIC ASPECTS

We summarize the main pipeline of CLANS in Algorithm 1. In a nutshell, there exist T tasks that need to be sequentially learned, and each task contains a training set and a validation set. We first train an initial network (or base architecture) $\mathcal{F}_1$ on $\mathcal{D}_1^{train}$, aiming to obtain the optimal parameters $\overline{\mathcal{W}}_1$. Meanwhile, we also initialize the gate operation and cumulative vector for the first task.

Subsequently, we perform the two procedures of current task learning and previous task refresh. Specifically, when a new task is coming, we first load Algorithm 2 to pick an optimal network architecture for the current task in an RL manner. After that, we load Algorithm 3 to make knowledge transfer from the new task to the previous one. We will refresh a previous task's architecture once we gain a better validation accuracy.

Algorithm 1: CLANS for Continual Learning.

Input: A sequence of tasks with training sets $\{\mathcal{D}_1^{train}, \mathcal{D}_2^{train}, ..., \mathcal{D}_T^{train}\}$, validation sets $\{\mathcal{V}_1^{valid}, \mathcal{V}_2^{valid}, ..., \mathcal{V}_T^{valid}\}$.
Output: Optimized network for each task: $\mathcal{F}_t = (\mathcal{N}_t, \overline{\mathcal{W}}_t)$ for $t = 1, 2, \ldots, T$.
Set $\mathcal{KP} = \emptyset$;
for $t = 1, 2, \ldots, T$ **do**
  if $t = 1$ **then**
    `// Train the first task`
    Let $\mathcal{KP}_1 = \mathcal{KP} \cup \mathcal{N}_1$;
    Initialize gate operation and cumulative vector as $\{\boldsymbol{g}_1^l = \mathbf{1}\}_{l=1}^L$ and $\{\boldsymbol{m}_l^{\leq 1} = \mathbf{0}\}_{l=1}^L$;
    Train the initial network $\mathcal{F}_1$ on $\mathcal{D}_1$ with the weights optimized as $\overline{\mathcal{W}}_1$;
    Add $\mathcal{V}_1^{valid}$ into $\mathcal{B}_1$;
  else
    `// REINFORCE for Current Task`
    Expand the network and obtain the trained $\overline{W}_t$ by Algorithm 2;
    Update $\mathcal{KP}_t$ and $\mathcal{B}_t$;
    `// Selection Refresh for Previous Tasks`
    Refresh each previous task architecture by Algorithm 3;

Algorithm 2: REINFORCE for Current Task.

Input: The training/validation set of current task $\mathcal{D}_t^{train}/\mathcal{D}_t^{valid}$; previous gate vectors $\{\boldsymbol{g}_i\}_{i=1}^{t-1}$; reply buffer $\mathcal{B}_{t-1}$; cumulative gate vector $\boldsymbol{m}_{\leq t-1}$.
Output: Optimized network $\mathcal{F}_t = (\mathcal{N}_t, \overline{\mathcal{W}}_t)$ and $\boldsymbol{m}_{\leq t}$.
for $j = 0, 1, \ldots, J$ **do**
  if $j = 0$ **then**
    Use $\mathcal{D}_t^{valid}$ and $\mathcal{B}_{t-1}$ to compute initial states (i.e., $\boldsymbol{s}_t^j$) via Eq.(4); ▷ task similarity
  else
    Use $\boldsymbol{g}_{t,j}$ and $\{\boldsymbol{g}_i\}_{i=1}^{t-1}$ to compute the states $\boldsymbol{s}_t^j$ based on Eq.(5); ▷ selection similarity
  Generate actions $\boldsymbol{a}_t^j = \{a_{t,j}^l\}_{i=1}^L$ from $\boldsymbol{s}_t^j$ according to Eq.(6) and Eq.(7);
  Configure the task network as $\mathcal{F}_t^j = \{\mathcal{N}_t^j, \mathcal{W}_t^j|_{\mathcal{W}_t^{old} = \overline{\mathcal{W}}_{t-1}^{old}}\}$ based on $\boldsymbol{a}_t^j = \{a_{t,j}^l\}_{i=1}^L$;
  Set $\boldsymbol{m}_{\leq t,j}^l = [\boldsymbol{m}_{\leq t-1}^l, \mathbf{0}(a_{t,j}^l)]$ for each layer l;
  Initialize the layer embeddings $\boldsymbol{e}_{t,j} = \{\boldsymbol{e}_{t,j}^l\}_{i=1}^L$ and obtain gate vectors $\boldsymbol{g}_{t,j} = \{\boldsymbol{g}_{t,j}^l\}_{l=1}^L$;
  Train the configured $\mathcal{F}_t^j = \{\mathcal{N}_t^j, \mathcal{W}_t^j|_{\mathcal{W}_t^{old} = \overline{\mathcal{W}}_{t-1}^{old}}\}$ with objective Eq.(1) based on Eq.(9) and Eq.(10);
  Compute the reward R_t^j via Eq.(8) and obtain $\boldsymbol{g}*_{t,j}$ according to Eq.(11);
  Set $\boldsymbol{g}_{t,j} = \boldsymbol{g}_{t,j}^*$; ▷ For similarity measurement
  Update the parameters θ in Neuron Expander;
Store the best network parameter configuration $\mathcal{F}_t = (\mathcal{N}_t, \overline{\mathcal{W}}_t)$ that has the greatest reward;
Update the final cumulative gate vector $\boldsymbol{m}_{\leq t}$ based on Eq.(3);

Algorithm 3: Selection Refresh for Previous Tasks.

Input: previous knowledge pool $\mathcal{KP}_{t-1}$; previous gate vectors $\{\boldsymbol{g}_i\}_{i=1}^{t-1}$; reply buffer $\mathcal{B}_{t-1}$; cumulative gate vector $\boldsymbol{m}_{\leq t-1}$.
Output: Refreshed network for each task: $\mathcal{F}_t = (\mathcal{N}_t, \overline{\mathcal{W}}_t)$ for $t = 1, 2, \ldots, t-1$.
i=t-1;
while $i > 0$ **do**
  Expand each layer embedding of task T_i through Eq.(12);
  Set new neuron selection via Eq.(13);
  Train the refreshed $\mathcal{F}_i'$ with objective Eq.(1) based on Eq.(9) and Eq.(10);
  Obtain new validation accuracy acc_i^{new} on $\mathcal{B}_{t-1}^i$;
  if $acc_i^{new} > acc_i^{old}$ **then**
    Let $\mathcal{F}_i = \mathcal{F}_i'$;
  else
    Withdraw the update;
  $i = i - 1$;
Return a refreshed network for each task T_i where $i = 0, 1, \cdots, t-1$;

B.2 Details of Experimental Setup

In this part, we describe our experimental setup in detail, including the datasets, evaluation protocols, and implementations.

B.2.1 Datasets

The original datasets used in our study are summarized in Table 4. In detail, we conducted the results on four variants, including PMNIST, RMNIST, CIFAR-100, and TinyImageNet. PMNIST and RMNIST are two variants of the original MNIST dataset [1] containing a large number of 28×28 monochrome images of handwritten digits. In addition, PMNIST and RMNIST are widely used in CL, where each task of the former is transformed by a fixed and different permutation of pixels, while each task of the latter is rotated by a different angle between 0 to 360 degrees. CIFAR-100 is a CIFAR object recognition datasets with 100 classes. We follow Rebuffi et al. (2017) and randomly divide CIFAR-100 into 10 tasks, where each task contains 10 different classes and their examples. TinyImageNet [2] contains 100,000 64×64 colored images in 200 classes. We split ten 20-way classification tasks from the original TinyImageNet for continual learning. For fairness, we randomly sample a subset of the original dataset and also make the validation set the same as the training set following Serra et al. (2018). For CIFAR-100 and TinyImageNet, we split 20% of the training set from each task for validation purposes.

Table 4: The statistics of four benchmark datasets.

Dataset	Train	Test	Classes
MNIST LeCun (1998)	60,000	10,000	10
CIFAR-100 Krizhevsky et al. (2009)	60,000	10,000	100
TinyImageNet Kang et al. (2022)	100,000	10,000	200

B.2.2 Details of Evaluation Metrics

To fairly show the model performance and the ability of knowledge consolidation (including back and forward knowledge transfer), we use three metrics, i.e., **ACC**, **BWT**, and **Trans**. ACC is a common metric to evaluate the CL performance. After all tasks are continually well learned, we calculate the average accuracy of all tasks, where the accuracy of each task, denoted by $acc_{T,t}$, is obtained by testing its corresponding test data. To measure backward knowledge transfer, BWT can show the impact of new learning tasks on the accuracy performance of old tasks. Furthermore,

[1] http://yann.lecun.com/exdb/mnist/
[2] https://www.kaggle.com/c/tiny-imagenet

BWT>0 indicates the learning of new tasks has a positive impact on old task performance, BWT<0 indicates that the learning of the new task has a negative knowledge transfer on the old task. When BWT is a large negative value, we say that the CL model confronts a *Catastrophic Forgetting* problem. If BWT$=0$, we say the CL model has no forgetting issue. To measure whether learned knowledge in old tasks facilitates the learning of new tasks, we use Trans to uncover the ability of forward knowledge transfer. Specifically, we treat each task training as a single task learning and compare its accuracy performance with the task performance in the CL context. More formally, these three metrics are defined as follows:

$$\textbf{Average Accuracy: } \mathrm{ACC} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{acc}_{T,t}; \tag{14}$$

$$\textbf{Backward Transfer: } \mathrm{BWT} = \frac{1}{T-1}\sum_{t=1}^{T-1} \mathrm{acc}_{T,t} - \mathrm{acc}_{t,t}; \tag{15}$$

$$\textbf{Forward Transfer: } \mathrm{Trans} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{acc}_{T,t} - \overline{\mathrm{acc}}_t. \tag{16}$$

Herein, $\overline{\mathrm{acc}}_t$ is the task t's accuracy performance in a single task learning manner and $acc_{t,t}$ is the accuracy of task t on the corresponding test data after it is well trained in the CL context.

B.2.3 Details of Experimental Setup

Base Architecture. We explain the architecture details of task networks including MLPs and CNNs. Note that our baselines also use the same architectures for a fair comparison.

- **MLP for PMNIST and RMNIST:** We follow Serra et al. (2018) and start with 784-2000-2000-10 neurons with RELU activation for the first task. For the remaining tasks, we only allow each task to scale up to 30 neurons per layer, which is similar to Xu & Zhu (2018).
- **CNN for CIFAR-100 and TinyImageNet:** We also follow Serra et al. (2018) and extend a modified version of AlexNet for the first task. In more detail, it has three convolutional layers with 64, 128 and 256 filters, with 4×4, 3×3, and 2×2 kernel sizes, respectively, and plus two fully-connected layers of 2048 neurons each. Also, we use rectified linear units as activation and utilize a 2×2 max-pooling operation after three convolutional layers. For the remaining tasks, we allow each task to add up to 30 filters per convolutional layer.

Baselines: We have used 10 CL methods for comparison with CLANS. SGD is the simplest method and we implemented it by ourselves. For the remaining baselines, we extend their publicly available source codes to conduct the experiments. The url's for the source codes are listed in Table 5.

Table 5: The public source codes of baselines.

Method	Source
EWC Kirkpatrick et al. (2017)	`https://github.com/joansj/hat/tree/master/src/approaches`
two variants of IMM Lee et al. (2017)	`https://github.com/joansj/hat/tree/master/src/approaches`
PGN Rusu et al. (2016)	`https://github.com/joansj/hat/tree/master/src/approaches`
DEN Yoon et al. (2018)	`https://github.com/jaehong31/DEN`
RCL Xu & Zhu (2018)	`https://github.com/xujinfan/Reinforced-Continual-Learning`
HAT Serra et al. (2018)	`https://github.com/joansj/hat`
SupSup Wortsman et al. (2020)	`https://github.com/RAIVNLab/supsup`
WSN Kang et al. (2022)	`https://github.com/ihaeyong/WSN`

Hyperparameter configurations. We implemented our CLANS in Python using Pytorch library and all the experiments ran on a single NIVDIA GTX1080 GPU. We trained all methods, including our CLANS, with SGD optimizer. We set the maximum number of iterations for reinforcement learning to 50; the hidden size of GRU to 100; λ is 0.75; the initial learning rate on four datasets was set to 0.05; and the number of training epochs for the task network on PMNIST, RMNIST, CIFAR-100, and TinyImageNet are 30, 30, 50, and 80, respectively. We note that the source code of CLANS is uploaded as Supplementary Materials.

C.3 Additional Results

C.3.1 Performance Deviations

To alleviate the influence of task mixture in our study, we shuffle the tasks with 5 different seeds, resulting in 5 lists with different task orders. Table 6 reports the model performance as a supplement, where the averages and standard deviations are provided.

Table 6: CL Performance on four datasets.

Model	P-MNIST			R-MNIST		
	ACC	BWT	Trans	ACC	BWT	Trans
SGD	81.37(±1.178)	-24.52 (±1.825)	-17.06(±1.178)	72.83(±0.123)	-25.32(±0.533)	-25.08(±0.123)
EWC	94.20(±0.705)	-0.32(±0.048)	-4.23(±0.705)	94.86(±0.048)	-0.73(±0.027)	-3.05(±0.048)
IMM-mean	80.10(±1.860)	-1.13(±0.117)	-18.33(±1.860)	88.81(±1.120)	-0.96(±0.215)	-9.10(±1.120)
IMM-mode	93.13(±0.546)	-4.17(±0.510)	-5.30(±0.546)	89.48(±0.342)	-7.40(±0.235)	-8.43(±0.342)
PGN	91.89(±0.264)	0.00(±0.000)	-6.54(±0.264)	90.01(±0.126)	0.00(±0.000)	-7.90(±0.126)
DEN	91.96(±1.258)	-0.41(±1.258)	-6.47(±1.258)	91.53(±0.241)	-0.52(±0.176)	-6.38(±0.241)
RCL	92.28(±0.410)	0.00(±0.000)	-6.15(±0.410)	93.97(±0.035)	0.00(±0.000)	-3.94(±0.035)
HAT	98.10(±1.184)	0.00(±0.000)	-0.33(±1.184)	97.89(±0.091)	0.00(±0.000)	-0.02(±0.091)
SupSup	96.32(±0.254)	0.00(±0.000)	-2.11(±0.254)	97.15(±0.782)	0.00(±0.000)	-0.73 (±0.782)
WSN	96.46(±0.374)	0.00(±0.000)	-1.97(±0.374)	97.32(±0.873)	0.00 (±0.000)	-0.59(±0.873)
CLANS	**98.21(±0.388)**	**0.01(±0.021)**	**-0.22(±0.388)**	**97.93(±0.262)**	**0.00(±0.000)**	**0.02(±0.262)**
Model	**CIFAR-100**			**TinyImageNet**		
	ACC	BWT	Trans	ACC	BWT	
SGD	57.86(±2.093)	-20.32(±1.006)	-15.77(±2.093)	13.60(±1.261)	-25.32(±1.137)	-25.14(±1.261)
EWC	68.12(±0.347)	-1.74(±1.154)	-5.51(±0.347)	35.24(±0.471)	-1.34(±0.245)	-2.50(±0.471)
IMM-mean	60.08(±0.028)	-0.72(±0.864)	-13.55(±0.028)	26.57(±0.818)	-2.89(±0.735)	-11.17(±0.818)
IMM-mode	61.50(±0.594)	-15.58(±1.805)	-12.13(±0.594)	27.05(±0.625)	-14.40(±0.511)	-10.69(±0.625)
PGN	58.28(±1.474)	0.00(±0.000)	-15.35(±1.474)	33.56(±0.654)	0.00(±0.000)	-4.18(±0.654)
DEN	59.32(±1.508)	-1.24(±1.391)	-12.79(±1.508)	33.86(±0.981)	-1.30(±0.513)	-3.88(±0.981)
RCL	61.07(±1.841)	0.00(±1.147)	-12.56(±1.841)	34.12(±0.696)	0.00(±0.000)	-3.62(±0.696)
HAT	70.67(±0.554)	0.00(±0.000)	-2.96(±0.554)	37.92(±0.462)	0.00(±0.000)	0.18(±0.462)
SupSup	68.98(±0.373)	0.00(±0.000)	-4.65(±0.373)	37.28(±0.271)	0.00(±0.000)	-0.46(±0.271)
WSN	69.34(±0.434)	0.00(±0.000)	-4.29(±0.434)	37.41(±0.312)	0.00(±0.000)	-0.33(±0.312)
CLANS	**71.41(±0.785)**	**0.02(±0.017)**	**-2.22(±0.785)**	**39.60(±0.210)**	**0.10(±0.034)**	**1.86(±0.210)**

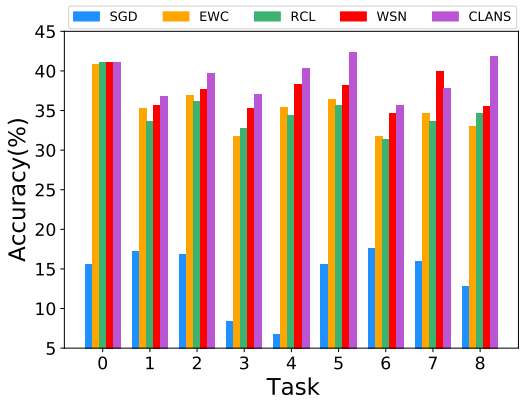


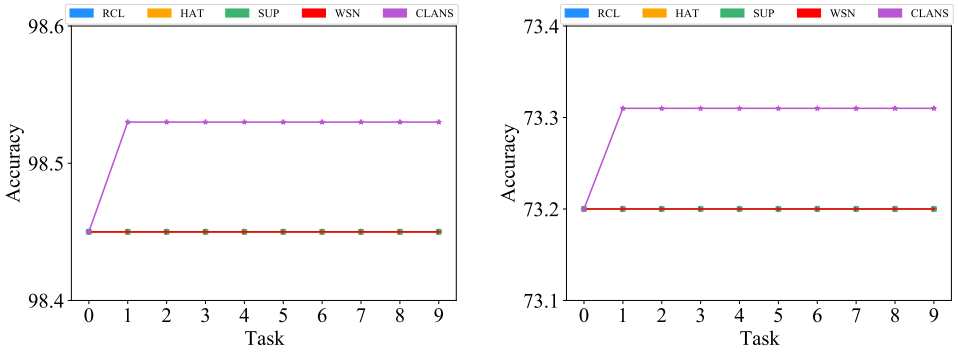
Figure 6: Accuracy performance for each task on TinyImageNet.

C.3.2 ACCURACY PERFORMANCE ON TINYIMAGENET

Due to the space limit, Fig. 2 does not report the results on TinyImageNet. After completing continual learning, we test the accuracy performance of each task on TinyImageNet. Fig. 6 shows that CLANS has competitive results against the baselines.

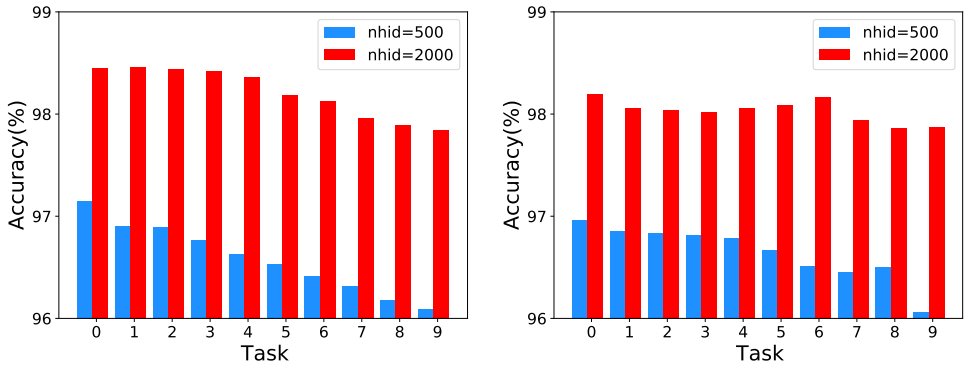
C.3.3 FORGETTING CURVE

Solving catastrophic forgetting (CF) is a key topic for continual learning. Regularization methods such as EWC and IMM usually confront a severe CF problem. Due to the freezing operation on used parameters, other network adaptation methods such as DEN, HAT, and WSN have no CF problem. Fig. 7 presents the changes of the test accuracy on the first task as more new tasks are sequentially learned. We clearly find that the baselines such as HAT and WSN fail to improve but maintain the test accuracy of the first task as more tasks are learned. We say they have no CF problem. In contrast, CLANS can even improve the accuracy gain on the first task as more tasks are learned.



(a) PMNIST (b) CIFAR-100

Figure 7: Accuracy performance on the first task as more tasks are sequentially learned.



(a) PMNIST (b) RMNIST

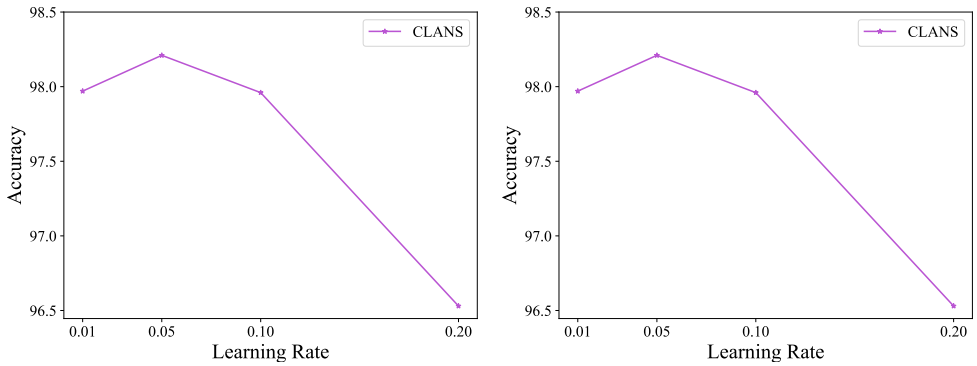
Figure 8: The impact of the base architecture.

D.4 SENSITIVITY ANALYSIS

We also investigate the potential impacts of hyperparameter settings in CLANS, including the hidden size of GRU, iteration number of RL, the scale of replay buffer, etc. The details are:

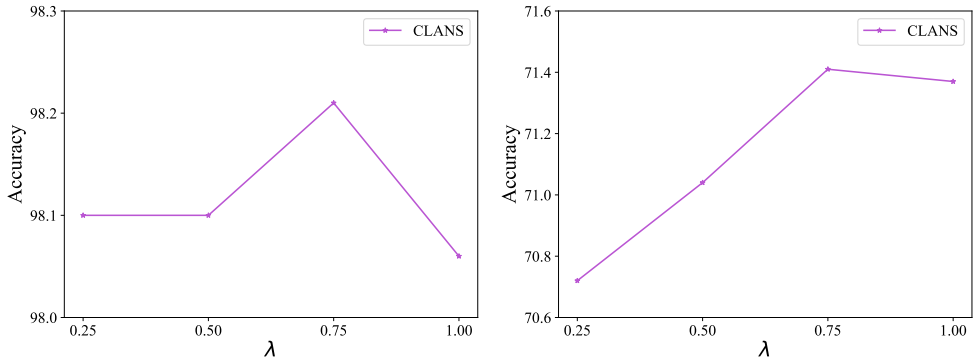
The Sensitivity of Base Architecture. Herein, we use a much smaller base architecture as the initial task network for the first task. Specifically, we design 784-500-500-10 neurons with RELU

activation for the first task of PMNIST and RMNIST, respectively. As Fig. 8 shows, we can observe the smaller base architecture brings worse performance, where 'nhid' refers to the number of neurons (filters) for the first task. We consider the plausible reason is that a smaller base architecture will give us a dense network that has limited search space in the expandable gating mechanism, which would hinder us to find the best network for new task learning. In addition, this phenomenon has also been observed in recent studies such as HAT Serra et al. (2018), Supsup Wortsman et al. (2020), and WSN Kang et al. (2022). More importantly, this trial stimulates us to provide a slightly large dense network as the base architecture of the first task in CLANS.



(a) PMNIST (b) CIFAR-100

Figure 9: The impact of the initial learning rate.



(a) PMNIST (b) CIFAR-100

Figure 10: The sensitivity of λ.

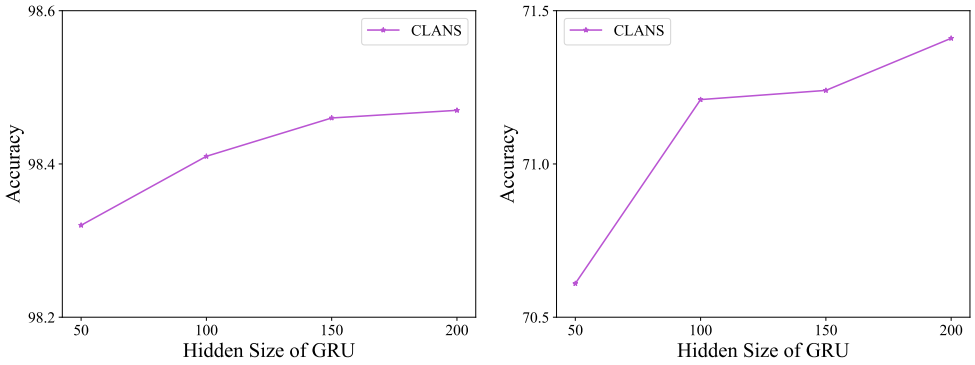
The Sensitivity of Initial Learning Rate. We also investigate the sensitivity of initial learning rate. As Fig. 9 shows, the task network in CLANS is sensitive to the initial learning rate.

The Sensitivity of λ. We investigate the sensitivity of hyper-parameter λ on gating operation. Fig. 10 shows that λ in CLANS is more sensitive to CIFAR-100 than PMNIST. And we should carefully choose a appropriate λ for each dataset.

The Sensitivity of hidden size of GRU. In addition, we also study the sensitivity of hidden size of GRU. Fig 11 shows, We can find that CIFAR-100 is more sensitive when the hidden size of GRU becomes gradually smaller.

E.5 Applications and Limitations

In practice, our CLANS can be applied to various applications, we here outline three potential applications of our CLANS.



(a) PMNIST (b) CIFAR-100

Figure 11: The sensitivity of hidden size of GRU.

- **Mobile Computing:** The advances in mobile devices and the Internet of Things (IOTs) enable increased use of mobile computing resources to solve multiple complex tasks. However, due to the limited memory and computing resources, it is unrealistic for mobile devices to download an extremely large network for task training. Hence, our CLANS with adaptive neuron selection can provide mobile devices with a more compact but effective network for task learning. Also, the communication cost between the mobile sides and the cloud will be reduced accordingly.
- **Federated Learning:** This emerging paradigm has a significant impact in information security, especially for the financial industry. Federated learning works between cloud centers and massive remote computing nodes. Each node can download the training model shared by other nodes from the cloud center, which usually leads to privacy leakage problems. Since CLANS tries to discover an optimal sub-network for each task learning, we believe that assigning such a sub-network to each computing node can mitigate the risk of privacy issues to some extent while such a sub-network enables maintaining the accuracy performance.
- **Network Compression (Pruning):** Over-parameterized network is a urgent concern in AI. In recent years, numerous deeper and wider neural networks have been designed to handle various domain sources such as documents and videos. Although there are massive studies that can make network compression to tackle the over-parameterized issue, they only concentrate on pruning the network to solve a single task, and they fail to make the compression more flexible when faced with multiple tasks in sequence. We are convinced that our work can provide a more open perspective for network compression to handle more flexible task scenarios.

As mentioned, CLANS adopts the merits of the network-adaption paradigm and makes each task network expandable and in order to control the expandability of the task network, we operate reinforcement learning to determine how many new neurons or filters can be added to each layer of the task network. In contrast to existing RL settings, our RL environment is conditioned on task similarity and selection similarity. Moreover, the neuron selection results are changed over iterations, which provides a dynamic signal to boost the Network Expander in CLANS to make a more accurate selection for the current task learning (cf. Fig. 3). To select useful old neurons from the earlier tasks, we can also use RL, however, it will consume a lot of time. Instead, we devise an expandable gating mechanism that can select useful neurons during task training. Due to these two profitable designs, CLANS enables the knowledge consolidation between old and new tasks, including back and forward knowledge transfer. However, we have following two limitations about CLANS.

- **Computational cost in RL.** Reinforcement learning is a more intelligent manner to mimic human cognitive processes. Although we only use RL to determine the network expansion, it still brings us much computational cost as well as additional memory cost. In the future, we consider whether we can first bridge the relation between the expansion needs and data

distribution, and devise a heuristic mechanism to decide how many new neurons should be added according to such a relation.

- **Similarity measurement & Replay Buffer.** As we use task similarity to investigate the correlations between the new task and earlier tasks, which will bring additional memory and computing costs. Besides, we refresh the task network of each previous task using the replay buffer, which also brings huge computing resources and replay bias. In our future work, we consider whether we can use generative models to decrease resource costs.

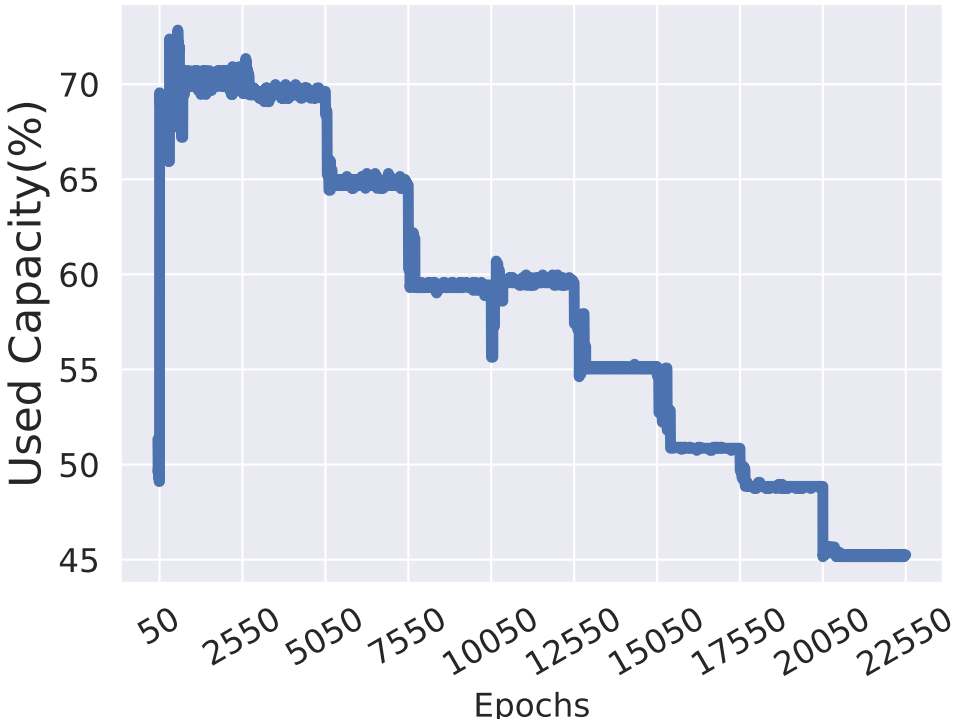
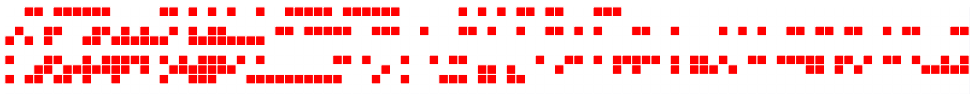


Figure 12: Network capacity on CIFAR-100 throughout continual learning.

F.6 Case study

In our case study, we first provide a macro perspective to show the capacity usage of the knowledge pool on CIFAR-100 throughout continual learning. As shown in Fig 12, we can find a significant increase in capacity usage of the knowledge pool when CLANS starts to receive new tasks. It is natural as the knowledge pool has no useful information for the coming tasks. As the number of training tasks increases, we find that capacity usage actually decreases. The plausible reason is that the knowledge pool has more useful knowledge extracted from old tasks, which can help the learning of future task learning.

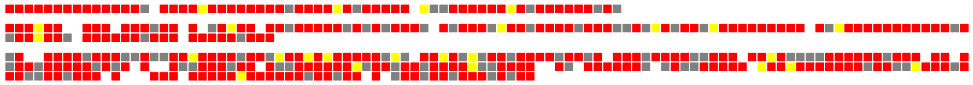
Additionally, we visualize the filters selected from the base architecture in knowledge pool for each task in CIFAR-100. As shown in Fig. 13, each square represents a filter of CNN. A white square means it has never been used by any task; a square in red represents that it was trained by some previous tasks and is now reused by the current task; a grey square indicates it has been used by a previous task but not reused by the current task; and the yellow squares depict newly added filters for the current task learning. As the task increases, CLANS drops many old filters and only reuses part of the used filters. Furthermore, we were surprised to find that the number of newly added filters decreases as the task increases, which implicitly indicates that CLANS prefers to learn a better architecture for the current task on earlier reused neurons than to expand the task network.



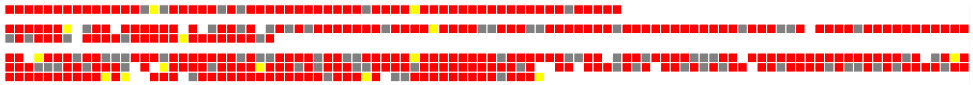
(a) Task 0

(b) Task 1

(c) Task 2



(d) Task 3



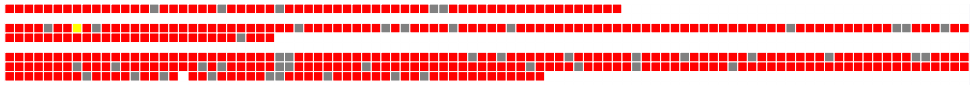
(e) Task 4

(f) Task 5

(g) Task 6

(h) Task 7

(i) Task 8



(j) Task 9

Figure 13: Examples of CNN architectures for CIFAR-100.