Relational Experience Replay: Continual Learning by Adaptively Tuning Task-Wise Relationship

Quanziang Wang, Renzhen Wang, Yuexiang Li, Dong Wei, Hong Wang, Kai Ma, Yefeng Zheng, Fellow, IEEE, and Deyu Meng, Member, IEEE

Abstract—Continual learning is a promising machine learning paradigm to learn new tasks while retaining previously learned knowledge over streaming training data. Till now, rehearsal-based methods, keeping a small part of data from old tasks as a memory buffer, have shown good performance in mitigating catastrophic forgetting for previously learned knowledge. However, most of these methods typically treat each new task equally, which may not adequately consider the relationship or similarity between old and new tasks. Furthermore, these methods commonly neglect sample importance in the continual training process and result in sub-optimal performance on certain tasks. To address this challenging problem, we propose Relational Experience Replay (RER), a bi-level learning framework, to adaptively tune task-wise relationships and sample importance within each task to achieve a better ‘stability’ and ‘plasticity’ trade-off. As such, the proposed method is capable of accumulating new knowledge while consolidating previously learned old knowledge during continual learning. Extensive experiments conducted on three benchmark image datasets (CIFAR-10, CIFAR-100, and Tiny ImageNet) and two text datasets (20News and DBpedia) show that the proposed method can consistently improve the performance of all baselines and surpass current state-of-the-art methods.

Index Terms—Continual learning, stability-plasticity dilemma, bi-level optimization.

I. INTRODUCTION

Deep neural networks have demonstrated remarkable performance across a wide range of machine learning tasks, including classification [1], [2], [3], semantic segmentation [4], [5], [6], and object detection [7], [8], [9]. Nonetheless, these models often face challenges in continual learning (CL) scenarios, where knowledge accumulates incrementally from data generated by a non-stationary distribution. Specifically, the models tend to overwrite previously acquired knowledge whenever new tasks come in, resulting in catastrophic forgetting [10], which poses a significant challenge in continual learning.

Recently, various CL methods have been proposed to alleviate the catastrophic forgetting problem. Existing algorithms for CL can be roughly divided into three categories [11]: 1) Model-based approaches [12], [13], [14], [15], [16], which dynamically modify the architecture of the model to handle the new task and use specific model structures for different tasks in the testing phase; 2) Regularization-based approaches [17], [18], [19], [20], [21], which commonly impose constraints to maintain the model parameters that are relatively important for previously learned tasks; 3) Rehearsal-based approaches [22], [23], [24], [25], [26], [27], [28], [29], [30], which store a small subset of examples from previously learned tasks in a memory buffer and replay them during the learning of new tasks. In this study, we mainly focus on exploring the potential of rehearsal-based methods, as they have shown relatively stable and effective performance in addressing the catastrophic forgetting in continual learning.

Rehearsal-based methods aim to strike an appropriate balance between learning new information and retaining previous knowledge, which is referred to as the ‘stability-plasticity’ dilemma [31]. Many rehearsal-based methods, such as Experience Replay (ER) [22], [23] and DER++ [26], primarily concentrate on leveraging stored information from previous tasks to mitigate model forgetting during the training of new tasks. However, these approaches have paid limited attention to modeling the intricate relationships between tasks. For example, the loss function of ER is the sum of the losses of incoming new tasks, which poses an empirical challenge to the learning process.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
A. Continual Learning

Rehearsal-based Methods: The primary mechanism underlying rehearsal-based methods [22], [23], [24], [25], [26], [27], [28], [29], [33] is using the information of data from old tasks to prevent forgetting while training new tasks. Specifically, these methods typically involve saving a portion of data samples from old tasks as a memory buffer, which is subsequently used alongside new incoming data samples to train the model. For example, A-GEM [34] relaxes the constraints in GEM by only restricting the dot product of the new and old sample gradients to be non-negative, so as to improve the computational efficiency of the algorithm. iCaRL [24] stores the most representative samples, i.e., located around the class center in the latent space as the memory buffer, and uses the Nearest Class Mean (NCM) classifier to mitigate the impact of feature representation changes. Rainbow Memory [28] constructs the memory buffer by sampling more representative samples, determined based on the prediction confidence of the data after applying various augmentations. In addition to the ground truth labels, DER++ [26] also saves data probabilities yielded by the model from previous epochs in the memory buffer as soft labels, which are used for distillation to further prevent forgetting. On the flip side, rehearsal-based methods commonly lead to class imbalance problems, as they rely on a small memory buffer to store a limited number of examples from old tasks. To address this problem, LUCIR [33] normalizes the predicted logits and imposes constraints on the features to correct the imbalance problem between new and old samples. ER-ACE [29] calculates the loss function of the new and old tasks independently to alleviate the mutual interference between them.

Authorized licensed use limited to: University of Gothenburg. Downloaded on September 20, 2023 at 13:17:51 UTC from IEEE Xplore. Restrictions apply.
Other Continual Learning Methods: Model-based approaches [12], [13], [14], [15], [16], [35] aim to enhance the model’s adaptability to new tasks by automatically modifying its architecture. Despite their impressive performance in simulation experiments, these methods are known to pose challenges in training and optimization, often requiring significant computational resources. For example, PNN [12] saves all networks of previous tasks to avoid forgetting, which often occupies a large memory buffer and needs to train another network for the new task. Regularization-based approaches [17], [18], [19], [20], [21], [36] design different constraints to prevent changing important parameters of previous tasks during training new tasks. Typically, EWC [17] limits the updating of important parameters, which are selected by the Fisher matrix from a Bayesian perspective. SI [19] determines important parameters by evaluating the impact of parameter changes on the loss function. LwF [18] saves previous model predictions of new task samples as soft labels to distill the extracted knowledge. Despite their simplicity and effectiveness, these approaches also face challenges such as hyper-parameter tuning and sensitivity to the choice of regularization parameters. Besides, continual learning is also applicable to various realistic problems, such as semantic segmentation [37], [38], [39], few-shot learning [40], [41], [42], and emotion detection [43], [44], etc. For more details, we recommend referring to the literature provided in [11], [45].

B. Sample Weighing Strategy

In terms of sample weighing, our method is closely related to L2RW [46] and Meta Weight Net (MW-Net) [47]. These approaches involve training a classification network on a noisy label dataset and being optimized through a meta-learning strategy to mitigate the impact of noisy labels. This meta-learning process is guided by a small clean dataset, referred to as the meta set. However, the challenges posed by CL are different from those of the noisy label problem. Specifically, the unstable data flow and severe catastrophic forgetting make it challenging to directly apply these sample weighing methods to CL. To mitigate this problem, we propose a novel pairwise sample weighing strategy to model task relationships and it does not require any additional high-quality data as the meta set like L2RW or MW-Net due to the limitation of continual learning settings. To our knowledge, this should be the first work to use the automatic sample weighing strategy for continual learning.

III. PRELIMINARIES

In this section, we briefly introduce the settings of the continual learning problem and two main rehearsal-based baselines. In continual learning, the model $f$ is required to learn a stream of tasks $T = \{D_1, D_2, \cdots \}$ and for each task $D_i$, only a few samples can be stored in a memory buffer $\mathcal{M}$, where the buffer size is denoted as $M$. Typically, the model $f : \mathcal{X} \rightarrow \mathbb{R}^{C_i}$ is a classification neural network parameterized by $\theta$, where $C_i$ denotes the total number of classes across tasks 1 to $t$. In this study, we mainly focus on the Class Incremental (Class-IL) and Task Incremental (Task-IL) settings of continual learning. Specifically, in Task-IL, the task boundary is clear, i.e., the task ID $t$ is known during both training and testing phases. Therefore, the model can feed the input data to the corresponding task-specific classifier using masks [26] based on its task ID. Conversely, Class-IL is more challenging as the task ID remains inaccessible.

A. Experience Replay (ER)

ER is a fundamental rehearsal-based approach that aims to mitigate the forgetting of previously learned tasks while learning new ones by replying to samples stored in the memory buffer. Specifically, during each task training, it samples a batch of data $D^i = \{(x^i_j, y^i_j)\}_{i=1}^B$ from the current $t$-th task $D_t$ and another batch of data $M^i = \{(x^M_j, y^M_j)\}_{j=1}^B$ from the memory buffer $\mathcal{M}$ with the same batch size $B$, where $x^i_j$ and $y^i_j$ represent an example and its corresponding label, respectively. The loss function of ER can be formulated as:

$$L^t(\theta) = \frac{1}{B} \sum_{i=1}^B \lambda^D_i L_{CE}(x^i_D; \theta) + \lambda^M_i L_{CE}(x^i_M; \theta), \quad (1)$$

where $L_{CE}$ denotes the cross-entropy (CE) loss, and $\lambda^D_i$ and $\lambda^M_i$ represent the sample weights. A prevalent convention is to set $\lambda^D_i = 1$ and $\lambda^M_i = \lambda$. The value of $\lambda$ is a crucial hyperparameter that necessitates manual specification before training in many previous works.

B. Sample Weights in Continual Learning

Considering the stochastic gradient descent (SGD) of (1), we can get the updating formulation as

$$\theta^{t+1} = \theta^t - \eta \frac{1}{B} \sum_{i=1}^B \lambda^D_i g^D_i + \lambda^M_i g^M_i, \quad (2)$$

where $\eta$ is the learning rate, and $g^D_i$ and $g^M_i$ are gradients of $x^D_i$ and $x^M_i$, respectively. From this equation, we can find that the weights $\lambda^D_i$ and $\lambda^M_i$ control the direction of gradient descent from two aspects. Firstly, as shown in Fig. 2(a), when the new task weights surpass those of the memory buffer, the total gradient will tend to descend more along the direction of old tasks, and vice versa. Moreover, for any different training samples $x_i$ and $x_j$, their sample weights also adjust the gradient direction as illustrated in Fig. 2(b). Consequently, proper sample weights can enhance the model training by effectively capturing the relationship between new and old tasks, as well as the sample importance within each class. Therefore, the fixed weights in

Fig. 2. Illustration of sample weights modeling (a) task relationships and (b) sample importance during training. Here, $g^D$ and $g^M$ represent gradients of new and old tasks, with $\lambda^D$ and $\lambda^M$ denoting their corresponding weights. Similarly, $g_1$ and $g_2$ are gradients of $x_i$ and $x_j$, with $\lambda_1$ and $\lambda_2$ denoting their corresponding weights.
prior studies might yield sub-optimal results and need further improvement.

IV. RELATIONAL EXPERIENCE REPLAY

Most existing works alleviate the model forgetting problem by storing more information about old tasks, potentially leading to increased storage requirements. In this study, we propose a novel method to enhance continual learning by focusing on an alternative perspective—assigning proper weights to training samples based on two primary factors: 1) task relationship between new and old tasks, and 2) sample importance within each task for network training. To account for the first factor, we consider that a sample from the memory buffer containing similar or semantically relevant information to the new task samples is more prone to suffer from disturbance, potentially leading to more forgetting. In this situation, the model should probably pay more attention to the memory buffer sample by assigning larger weights to this old sample or smaller weights to new task samples to alleviate the forgetting issue. Conversely, the model should appropriately assign larger weights to the new task samples to effectively learn new knowledge without disturbing old tasks too much. As for the second factor, the data points from the same class also tend to make different contributions to model training. The easy-to-learn samples deliver more representative feature information of their classes, while the ‘hard’ samples are more conducive to refining the classification boundaries.

As mentioned above, the loss weights \( \Lambda \), which control the balance between ‘stability’ and ‘plasticity’ during training across different continual learning scenarios, should be dynamically assigned to different samples instead of being manually tuned and fixed [26]. To this end, we propose a Relation Replay Net (RRN) that extracts the interaction knowledge of the new and old samples and generates their corresponding weights \( \Lambda \) dynamically. It facilitates the main classification network (dubbed Main Net) to achieve a better trade-off between ‘stability’ and ‘plasticity’. Intuitively, RRN depends on the training state of the Main Net, and the sample weights generated by RRN in turn affect the training of the Main Net. Therefore, instead of the naive end-to-end training, we adopt a bi-level learning framework to jointly optimize both RRN and the Main Net. The effectiveness of this bi-level optimization will be demonstrated in Section V-D. For simplicity, we initially apply the proposed approach to ER (termed Relational Experience Replay, or RER), and the implementation details based on more baselines can be found in Appendix B-A.

A. Overview

The RER framework, as depicted in Fig. 3, consists of two key components: Main Net \( f(\cdot; \theta) \) responsible for continual learning, which can be any commonly used backbone architecture, and RRN \( h(\cdot; \phi) \) utilized for assigning sample weights.

On the one hand, a CL model must be able to adapt to different new tasks based on the knowledge learned from the old tasks, i.e., task relationship would affect the ‘plasticity’ as aforementioned. Furthermore, each new task sample has a varying impact on the model’s plasticity, highlighting the importance of sample weights in fine-tuning the model’s attention. To model this dynamic ‘plasticity’, we assign weights for pairwise training samples from new and old tasks to train the Main Net, where the weights are generated by the RRN based on their relationships. Concretely, we combine the new task batch \( B^D \) and the memory batch \( B^M \) to pairwisely construct training sample batch \( B^{tr} = \{ (x_i^D, x_i^M) \}_{i=1}^B \), where we omit the labels for notation convenience. Then, the weighted loss function for the Main Net \( f \) formulates the inner loop optimization problem of the bi-level learning framework, that is:

\[
\theta^*(\phi) = \arg \min_{\theta} L^{tr}(B^{tr}; \theta, \phi)
\]

\[
\triangleq \frac{1}{B} \sum_{i=1}^B \lambda_i^D(\phi)L^{tr}(x_i^D; \theta) + \lambda_i^M(\phi)L^{tr}(x_i^M; \theta),
\]

(3)
where $L_{tr}$ is CE loss based on ER, and the sample weights
\[ \Lambda = \{ \lambda_i^D, \lambda_i^M \}_{i=1}^B \] of these data pairs are generated by the RRN automatically.

On the other hand, the model should possibly alleviate the forgetting issue while learning new tasks. The proposed RRN should pay more attention to ‘stability’ to prevent the Main Net from focusing too much on new tasks. We thus formulate the outer loop optimization problem over memory buffer data, making the Main Net returned by optimizing the inner loop one in (3) acts as a stable consolidation of knowledge from the learned tasks \(^1\), i.e.,
\[
\phi^* = \arg \min_\phi L_{bf}^{bf} \left( B_{bf}; \theta^*(\phi) \right)
\]
\[
\triangleq \frac{1}{B} \sum_{i=1}^B L_{bf}^{bf} \left( x_i; \theta^*(\phi) \right),
\]
where the $L_{bf}^{bf} (x; \theta)$ can be simply adopted as CE loss, and $B_{bf}$ is another batch of samples from the memory buffer $\mathcal{M}$, which is different from the inner-loop training data $B$. By dynamically generating specific weights for different samples, the RRN can help the Main Net to better balance the trade-off between new and old tasks, thereby facilitating the continual learning process.

Furthermore, the design of the RRN should consider the following two factors: 1) the RRN should take the \textit{interaction information} between the new and old tasks into account; 2) the input of the RRN should be \textit{’information abundant’} to ensure that the RRN could extract useful knowledge to generate meaningful weights. With this aim, we propose to pair each new task sample with a corresponding sample from the memory buffer and use their respective abstract information as inputs to the RRN. Specifically, for each sample pair $x_i^D$ from the new task and $x_i^M$ from the memory buffer, we take their losses $L_{tr}^{D,M} = [L_{tr}^D(x_i^D), L_{tr}^M(x_i^M)]$, and logit norm $\|z_{D,M}\|_2 = [\|z_i^D\|_2, \|z_i^M\|_2]$ as the inputs of the RRN, where $z = f(x; \theta)$ is the predicted logits. By taking into account both label and feature information, the RRN can capture the semantic and distributional similarities between the paired samples and generate appropriate weights for the Main Net to balance the importance of new and old tasks during training. Thus, the paired sample weights generated by the RRN can be formulated as:
\[
[\lambda_i^D(\phi), \lambda_i^M(\phi)] = h \left( L_{tr}^{D,M}, \|z_{D,M}\|_2; \phi \right).
\]

In summary, the proposed RER forms a bi-level learning framework to simultaneously model the ‘plasticity’ and ‘stability’ in the inner and outer loop optimization problems, respectively. The RRN, guided by a relatively balanced set $\mathcal{M}$, automatically generates sample weights to refine the optimization direction of the Main Net, thereby facilitating enhanced Main Net optimization and achieving a more favorable balance between the new and old tasks. In the inference stage, we can directly predict the testing images by the Main Net without involving the RRN. Note that our method can be easily adapted to other \textit{rehearsal-based} baselines, such as ER-ACE [29] and DER++ [26] by some simple modification (please refer to Section V-C and Appendix B-A).

\section{Optimization Procedure}

Since it is difficult to find closed-form solutions, the optimization of $\theta$ and $\phi$ as shown in (3) and (4) depends on two nested loops, which is computationally expensive. Considering computational efficiency and the large scale of data to be processed, we adopt an alternative online gradient-based optimization strategy to solve the proposed bi-level learning framework.

\textbf{Updating $\theta$:} Referring to (3), given the parameter $\phi^k$ of RRN at iteration step $k$, we optimize the parameter $\theta$ of Main Net by one-step gradient descent:
\[
\theta^{k+1}(\phi) = \theta^k - \eta_\theta \nabla_\theta L_{tr}^{bf} \left( B_{bf}; \theta^k, \phi^k \right),
\]
where $\eta_\theta$ is the inner-loop learning rate. Note that the updated parameter $\theta^{k+1}(\phi)$ is actually a function of $\phi$.

\textbf{Updating $\phi$:} With Main Net parameter $\theta$, we can optimize RRN parameter $\phi$ by (4) given $\theta^{k+1}(\phi)$ by the following formulation:
\[
\phi^{k+1} = \phi^k - \eta_\phi \nabla_\phi L_{bf}^{bf} \left( B_{bf}; \theta^{k+1}(\phi^k) \right),
\]
where $\eta_\phi$ is the outer-loop learning rate. More details of the gradient calculation can be found in Appendix A.

\section{Theoretical Analysis}

According to (6) and (7), we have the following proposition to further reveal how the proposed method models the task-wise relationship.

\textbf{Proposition 1:} Let $g_{bf}^{bf} (x) = \frac{\partial L_{bf}^{bf}(x; \theta)}{\partial \theta} \bigg|_{\theta^k}$ and $g_{tr}^{fr} (x) = \frac{\partial L_{tr}^{fr}(x; \theta)}{\partial \theta} \bigg|_{\theta^k}$ denote the gradients of the buffer sample and the training sample with respect to the parameter $\theta$, respectively.

\begin{algorithm}
\caption{Relational Experience Replay Training Algorithm.}
\begin{algorithmic}[1]
\Require new task data $D_t$, memory buffer $\mathcal{M}$
\Ensure Main Net and Relation Replay Net (RRN) parameters $\{\theta, \phi\}$
\While {\textbf{new} $D_t \neq \emptyset$}
\While {$k < I_{\text{iter max}}$}
\State Sample a new task batch $B^D \in D_t$ and a buffer batch $B^M \in \mathcal{M}$
\State Construct a paired training batch $B_{\text{tr}} \leftarrow B^D$ and $B^M$
\State Calculate the inner-loop loss by (3)
\State Update $\theta^k$ by (6)
\State Sample another buffer batch $B_{bf} \in \mathcal{M}$
\State Calculate the outer-loop loss by (4)
\State Update $\phi^k$ by (7)
\State $k = k + 1$
\EndWhile
\EndWhile
\end{algorithmic}
\end{algorithm}

\(^1\)Note that the parameter $\phi$ of the Relation Replay Net is regarded as a hyper-parameter of that of the Main Net.
Then the updating formulation of \( \phi \) presented in (7) can be reformulated as

\[
\phi^{k+1} = \phi^k + \frac{\eta \eta}{B} \sum_{j=1}^{B} G(j) \cdot \left. \frac{\partial h_j(\phi)}{\partial \phi} \right|_{\phi^k},
\]

where the gradient \( \frac{\partial h_j(\phi)}{\partial \phi} = \left[ \frac{\partial \phi^D_j(\phi)}{\partial \phi}, \frac{\partial \phi^M_j(\phi)}{\partial \phi} \right]^T \), and the coefficient \( G(j) \) is

\[
G(j) = \frac{1}{B} \sum_{c=1}^{C_t} \left( \sum_{i=1}^{B_c} y^{b_f}(x_i) \right) \left[ g^{tr}(x^D_j), g^{tr}(x^M_j) \right].
\]

In Proposition 1, we demonstrate that the task-wise relationship in RRN can be implicitly modeled through the SGD updating process. This is achieved by computing the coefficient \( G(j) \), which represents the mean dot product similarity between the gradient of buffer samples and that of training samples within each class. This measure effectively captures the relationship between new tasks and previously seen classes from a gradient perspective. The proof can be found in Appendix A.

D. Implementation Details

**The Architecture of the Relation Replay Net:** We propose a two-hidden-layer neural network as the RRN. Intuitively, the first layer comprises two distinct linear layers designed to extract interaction information from paired training samples: one focuses on loss values, while the other attends to logit norms. Subsequently, the second layer is also a linear layer that merges the extracted information to automatically generate weights for the sample pairs. Fig. 4 illustrates the architecture of RRN where we set the number of hidden units as 16 for computation efficiency.

**Training Details:** When training a new task, the RRN aims to generate reasonable sample weights for both new task samples and buffer samples to improve the trade-off between ‘stability’ and ‘plasticity’, which needs a warm-up stage to explore the relationship between the new and old tasks. Specifically, in the warm-up stage (stage length denoted as \( \text{Iter}_{\text{warm}} \)), we update the RRN by (7) and update the Main Net via preset fixed weights like the previous methods. After the warm-up stage, we turn to use the sample weights generated by the RRN to guide the training of Main Net rather than prefixed weights. In addition, to reduce the calculation burden, we only update the RRN once while updating the Main Net for multiple steps (step number denoted as Interval). The specific choices of \( \text{Iter}_{\text{warm}} \) and Interval are presented in Section V-E.

V. EXPERIMENTAL RESULTS

To validate the effectiveness of our method, we conduct extensive experiments on various publicly available benchmark datasets for both image and text recognition tasks, including three image datasets, i.e., CIFAR-10, CIFAR-100 [48] and Tiny ImageNet [49], and two text datasets, i.e., 20News [50] and DBpedia [51]. Additionally, we present a thorough ablation analysis in this section to gain insight into our method.

A. Experimental Settings

For the three image datasets, we divide them into task sequences of different lengths. Specifically, CIFAR-10 is divided into five tasks with each task constituting a binary classification problem. Both CIFAR-100 and Tiny ImageNet datasets are divided into 10 tasks, where each task is a 1-of-10 and 1-of-20 classification problem, respectively. To ensure robust and reliable evaluation, the framework was trained for 50 epochs on each task across these three datasets following DER++ [26].

For the two text datasets, we follow the basic settings in [52]. We split 20News into 5 tasks, with each task consists of 4 classes. Utilizing a two-hidden-layer MLP as the classification model, we train each task for 200 epochs. Additionally, for DBpedia, we partition it into 7 tasks and each of them is a binary classification problem. We leverage a fixed pretrained BERT model to extract text embeddings, followed by training a linear classifier on these embeddings, and we train each task for 50 epochs.

As aforementioned, this study mainly focuses on the configurations of Class-IL and Task-IL. The detailed experimental settings are consistent with DER++ [26] for a fair comparison. In our experiments, we evaluate the model by two common-used metrics, i.e., Average Accuracy (ACC) and Backward Transfer (BWT) [25], [26]. We repeat each experiment five times to reduce the randomness of network training. Similar to [26], the total size of the memory buffer remains constant throughout the entire training process.

The proposed method is implemented with PyTorch [53]. The backbone of Main Net is the widely-used ResNet-18 [2]. In the inner loop, the Main Net is optimized by SGD with an initial learning rate of 0.03 for all datasets, and in the outer loop, Adam [54] is adopted to optimize RRN, where the initial learning rate is set as 0.001 with a weight decay of \( 10^{-4} \).

B. Comparison With State-of-the-art Methods on Image Classification

As previously mentioned, our proposed method aims to enhance general rehearsal-based baselines by taking into account the ‘stability-plasticity’ dilemma, which involves the relationship between new and old tasks, and the specific importance of different samples. We integrate this approach into three representative baselines: ER, ER-ACE [29], and DER++ [26], which are
Table I

| Buffer Size | Method          | CIFAR-10 | Tiny-ImageNet |
|-------------|-----------------|----------|--------------|
|             | ACC | BWT | ACC | BWT | ACC | BWT | ACC | BWT |
| Upper bound |     |     |     |      |     |      |     |      |
| Lower bound |     |     |     |      |     |      |     |      |
| 200         |     |     |     |      |     |      |     |      |
| GEM         | 28.54 ± 0.10 | 2.52   | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 |
| A-GEM       | 28.54 ± 0.10 | 2.52   | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 |
| iCaRL       | 28.54 ± 0.10 | 2.52   | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 |
| GSS         | 28.54 ± 0.10 | 2.52   | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 | 28.08 ± 0.10 | 2.52 |

For Tiny-ImageNet dataset, our method also consistently achieves the best results under almost all settings. Although iCaRL achieves sound BWT metric, it falls short in terms of ACC, revealing that a model that pays much attention to avoid forgetting may negatively impact classification performance. In contrast, our method improves both ACC and BWT compared to the corresponding baselines, highlighting that our method can achieve a better trade-off between new and old tasks. Notably, the results of GEM and GSS are not reported in Table I since the excessive computational overhead is unacceptable.

Moreover, we validate the effectiveness of our method on CIFAR-100 in Table II. Obviously, our method can achieve a significant improvement for all these three baselines across different buffer sizes. For instance, in the Class-IL setting with a buffer size of 100, our method applied to ER, RER-ACE, and DER can improve their ACC by 2.63%, 1.61%, and 5.81%, respectively. These results further demonstrate the strong adaptability of our method to multiple datasets and diverse settings.

C. Comparison on Text Classification

To investigate the scalability of our proposed method across various data modalities, we conduct experiments in the domain of text classification on 20News and DBpedia following [52]. The results are shown in Table III. It can be observed that our

termed Relational-ER (RER), Relational-ER-ACE (RER-ACE), and Relational-DER (RDER), respectively. Implementation details can be found in Appendix B-A.

Table I presents the comparison results between our proposed algorithm applied on the three baselines and various state-of-the-art methods over ACC and BWT metrics. These comparison methods include three regularization-based methods: eWAC [20], SI [19], and LwF [18], and four rehearsal-based methods: GEM [25], A-GEM [34], iCaRL [24], and GSS [55]. Besides, we also provide an upper-bound method and a lower-bound method for better reference, where the former trained on all data from old and new tasks together and the latter directly trained on the new task without any strategies to prevent model forgetting.

For CIFAR-10, it can be observed that our proposed approach can be adapted to different rehearsal-based baselines and achieve consistent performance improvement. For example, the proposed RDER achieves as much as 3.08% absolute performance gain compared to the baseline DER++ with a memory buffer of 200 under Class-IL. Besides, our method achieves significantly higher classification accuracy than all comparison state-of-the-art methods under different settings. On the other hand, our method can also significantly reduce the BWT of baselines, indicating that the proposed method can effectively reduce model forgetting while improving classification accuracy, i.e., achieving a better balance of ‘stability’ and ‘plasticity’.
method consistently improves the corresponding baseline under all settings. For 20News, the forgetting is more serious but our method can largely improve the baselines. Especially for DER++, our RDER can improve ACC by about 6.8% and increase BWT by about 14.8% with 200 buffer samples. Additionally, for another dataset DBpedia, our method can also significantly improve the performance of all three baselines for both ACC and BWT with various buffer sizes. These results indicate that our method can handle diverse data types and consistently improve the corresponding baseline by achieving better stability and plasticity trade-offs.

D. Discussion

To further explore the proposed method, we conduct more experiments and analyses in this section.

**Does Our Method Mitigate the Model Forgetting?** In order to better analyze the forgetting problem for old tasks, we visualize the accuracy change for each task during the continual learning process in Fig. 5 and Appendix B-C. The visualization results show that our method better mitigates forgetting for previous tasks, *i.e.*, consolidating knowledge for old tasks better than corresponding baselines. Note that the performance of RDER is slightly lower than DER++ for some new tasks (shown in Fig. 5(c)). This is because our method aims to improve the generalization of all tasks, rather than just paying attention to new tasks with more data.

**How Task Similarity Affects Continual Learning Models?** In this section, we aim to explore how task similarity affects model training and how our RRN captures task relationships to assign sample weights. To this end, we design two distinct setups using CIFAR-10 to simulate scenarios involving similar and dissimilar tasks in continual learning, as summarized in Table IV. Specifically, in Setup 1 and Setup 2, task 1 contains the same classes (‘ship’ and ‘truck’), while task 2 varies between the two setups: in Setup 1, it comprises semantically relevant classes (‘airplane’ and ‘automobile’) against task 1, whereas in Setup 2, it consists of semantically irrelevant classes (‘cat’ and ‘horse’). We then compare the baseline ER with our RER on these two setups and present the average accuracy after training each task in Table IV. We can observe that: 1) After training task 1, the performance of our method is comparable to that of the other methods under the same setting.
of the baseline method, as there is no need to consolidate old knowledge during this training stage. 2) After training task 2, the performance of the baseline model ER in Setup 1 is significantly lower than that in Setup 2. This is because the objects of the two tasks in Setup 1 are relatively easy to confuse, leading to more severe forgetting. Conversely, in Setup 2, the objects of task 2 differ from task 1, which makes them prone to confusion or misclassification, thereby increasing the risk of forgetting. Consequently, RRN prioritizes retaining memory buffer samples to alleviate forgetting and assigns them larger weights accordingly. Conversely, in Setup 2 as shown in Fig. 6(c) and (d), the sample weights of buffer samples are mainly distributed in the range of [0, 0.5], notably smaller than those in Setup 1. Meanwhile, the weights assigned to new samples are relatively larger than those in Setup 1. This is probably because RRN responds to the fact that new knowledge may not significantly affect the model of previously learned knowledge in subsequent epochs.

In summary, task relationships may significantly impact model training in continual learning, and our proposed RRN holds great potential to effectively capture such relationships by generating appropriate sample weights.

Why We Need the Bi-level Optimization? To evaluate the effectiveness of the bi-level optimization paradigm, we conduct an ablation study by end-to-end training of the Main Net and the RRN together based on DER++ (referred to as Vanilla). Specifically, we employ the same RRN to generate the paired sample weights and update the Main Net and the RRN by a single backward step through $L_{tr} + L_{bf}$. In Table V, we observe that the sample weights generated by the Vanilla method appear to be uninformative, and even impair the performance of the baseline approach (DER++). In contrast, our RDER approach consistently outperforms both DER++ and the Vanilla method, thereby demonstrating the effectiveness of the bi-level optimization paradigm.

| Memory Size | Method | Class-II | Task-II | Class-II | Task-II |
|-------------|--------|----------|---------|----------|---------|
| 200         | DER++  [26] | 62.30 ± 1.07 | 90.74 ± 1.01 | 62.85 ± 2.90 | 91.20 ± 1.88 |
|             | Vanilla | 62.53 ± 0.66 | 91.36 ± 0.77 | 65.38 ± 0.42 | 91.67 ± 0.80 |
| 500         | DER++  [26] | 72.11 ± 1.41 | 94.21 ± 0.32 | 72.57 ± 0.73 | 93.77 ± 0.33 |
|             | Vanilla | 73.99 ± 1.03 | 94.04 ± 0.43 | 85.56 ± 0.38 | 96.21 ± 0.22 |
| 5120        | DER++  [26] | 84.50 ± 0.63 | 95.91 ± 0.57 | 85.35 ± 0.24 | 96.19 ± 0.46 |
|             | Vanilla | 82.26 ± 2.45 | 95.3 ± 0.72 | 85.56 ± 0.38 | 96.21 ± 0.22 |

The bold values represent the best values compared with other methods under the same setting.

Table V.

ACC on the CIFAR-10 Dataset in Some Different Settings to Verify the Effectiveness of Our Method Design

Fig. 6. Weight distributions at epoch 45 during training for (a) buffer samples and (b) new task samples in Setup 1, (c) buffer samples, and (d) new task samples in Setup 2.

2For example, in Setup 1, the sample weight pair for new and old tasks are [ship, airplane], [ship, automobile], [truck, airplane], and [truck, automobile] as shown in Fig. 6(a) and (b).
the proposed bi-level optimization framework to achieve a better trade-off between ‘stability’ and ‘plasticity’.

**Why Use the Memory Buffer to Train Relation Replay Net in the Outer Loop?** In (4) of the outer-loop optimization, we utilize the memory buffer $\mathcal{M}$ to train the RRN. However, there are concerns regarding the risk of overfitting when using the memory buffer to guide the RRN. To address this issue, we investigate two approaches for constructing the outer-loop training set: 1) Splitting the memory buffer $\mathcal{M}$ into two sets, which are utilized for training the Main Net in the inner loop and the RRN in the outer loop, respectively [56]. 2) Alternatively, training the RRN in the outer loop using the entire memory buffer $\mathcal{M}$, as we propose.

We present a comparison between the two approaches based on DER++ and report the results in Table V, where the two methods are denoted as ‘Split RDER’ and ‘RDER’, respectively. Split RDER divides the memory buffer into two sets for outer- and inner-loop training with a ratio of $20\% - 80\%$, following [56], whereas the RDER approach does not involve such a split. The results in Table V indicate that Split RDER exhibits a slight improvement over the baseline DER++, but its performance is still lower than our proposed RDER. These findings suggest that the first approach may lead to better generalization of the RRN while reducing the number of memory samples used to train the Main Net. However, this reduction in training samples may create a more serious imbalance between the new and old tasks, thereby reducing the overall performance, particularly for limited buffer size.

### E. Ablation Study

In this section, we first validate the effect of the small buffer sizes on our method based on the three baselines. Then, we conduct a detailed ablation study based on RDER to evaluate the influence caused by two critical hyperparameters $\text{Iter}_{\text{warm}}$ and $\text{Interval}$ on CIFAR-10 with different buffer sizes.

#### Effect of Small Buffer Size:
To further investigate the impact of small buffer sizes on our method, we evaluate our method applied to the three baselines on CIFAR-10 in Table VI. Specifically, RDER enhances DER++ by about $+3.05\%$ with a buffer size of 50 under the Class-IL setting, demonstrating that our method can further explore the information in the memory buffer to enhance the Main Net overall performance.

#### Impact of the Warm-up Stage ($\text{Iter}_{\text{warm}}$):
Table VII presents the evaluation results for the various $\text{Iter}_{\text{warm}}$ under three different settings $[1, \frac{1}{2}, \frac{1}{3}] \times \text{Iter}_{\text{max}}$. The performance of RDER deteriorates significantly when a small value of $\text{Iter}_{\text{warm}}$ (i.e., 17) is used, suggesting that the RRN requires an adequate number of warm-up steps to generate meaningful sample weights to accurately capture the task-wise relationship and sample importance within each task. On the other hand, it can be observed a slight drop in performance under $\text{Iter}_{\text{warm}} = 33$, since the preset weights may mislead the training in the warm-up stage.
to be half of the number of iteration epochs for each task, which performs the best in Table VII.

Impact of the Relation Replay Net Updating Interval (Interval): Here we vary the value of Interval to investigate its impact on our framework. As shown in Table VIII, the setting Interval = 5 yields the highest classification accuracy across a range of memory buffer sizes. However, small Interval, such as Interval = 1, often results in decreased performance due to the frequent alternation between updating the Main Net and the RRN, leading to oscillation during training. Additionally, frequent updates of the RRN are computationally expensive, which can impede the convergence of the overall framework. Conversely, large Interval, such as Interval = 10, can lead to faster computation, but we observed a significant drop in performance due to inadequate training of the RRN, which can result in the generation of suboptimal sample weights. To strike a balance between accuracy and computational efficiency, we propose an empirical formulation Interval = #epoch/10, where #epoch represents the number of iteration epochs for each task.

Inputs of the Relation Replay Net: To further validate the effectiveness of the paired data and input terms of RRN, we conduct an ablation study based on RDER on CIFAR-10 with different memory buffer sizes. The results (final average accuracy) are listed in Table IX. We can observe that: 1) RDER-1, the model using only the loss of samples as input for RRN without incorporating the paired data structure, consistently improves baseline performance across diverse buffer sizes; 2) RDER-2, the model augmenting the RRN inputs of RDER-1 with the norm of predicted logits, further improves the performance over both ACC and BWT, indicating the efficacy of our proposed RRN input design; 3) RDER (ours), the model proposed in the main text, achieves the best performance and shows that the proposed paired data structure plays an essential role in the proposed RRN.

VI. CONCLUSION

In this paper, we focus on the ‘stability-plasticity’ dilemma in continual learning and strive to adaptively tune the relationship across different tasks and samples. To this end, we propose a novel continual learning framework, Relational Experience Replay, which pairwisely adjusts the sample weights of samples from new tasks and the memory buffer. The sample weights generated by the Relation Replay Net can facilitate the optimization of the Main Net to achieve a better trade-off between ‘stability’ and ‘plasticity’. The proposed method can be easily integrated with multiple rehearsal-based methods to achieve significant improvements. We theoretically and experimentally verify that the generated sample weights can extract the relationship between new and old tasks to automatically adjust the Main Net training and enhance the overall performance. We expect that our method can provide more insights into the ‘stability-plasticity’ dilemma and promote the development of the field of continual learning.

APPENDIX A

CALCULATION DETAILS ABOUT THE RELATION REPLAY NET UPDATING

In this section, we provide a detailed calculation of the derivatives of the RRN. Referring back to Section IV-B, the gradient descent of the RRN parameters is given in (7) and repeated here as

$$\phi^{k+1} = \phi^k - \eta_\phi \nabla_\phi L^{bf}(B^{bf}; \theta(\phi)),$$

(10)

where the $\theta(\phi)$ is the one-step updated parameters generated by (6).

Here we can use the chain rule to calculate the derivative of $\phi$ as follows.

$$\nabla_\phi L^{bf}(B^{bf}; \theta^k(\phi))$$

$$= \frac{1}{B} \sum_{i=1}^{B} \left( \frac{\partial L^{bf}(x_i; \theta)}{\partial \theta} |_{\theta^k} \right) _{\phi^k},$$

(11)

where the second term can be represented as

$$\frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k} = - \frac{\eta_\theta}{B} \sum_{j=1}^{B} \left( \nabla_\phi L^{tr}(x^D_j; \theta^k), \frac{\partial L^{tr}(x^D_j; \theta)}{\partial \theta} |_{\theta^k} \right) _{\phi^k}$$

$$+ \frac{1}{B} \sum_{i=1}^{B} \left( \frac{\partial L^{bf}(x_i; \theta)}{\partial \theta} |_{\theta^k} \right) _{\phi^k} \frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k}.$$}

(12)

Then substituting (12) into (11) and exchanging the order of the two summations, we can get

$$\nabla_\phi L^{bf}(B^{bf}; \theta^k(\phi))$$

$$= - \frac{\eta_\theta}{B} \sum_{j=1}^{B} \left( \frac{\partial L^{tr}(x^D_j; \theta)}{\partial \theta} |_{\theta^k} \frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k} \right) _{\phi^k}$$

$$+ \frac{1}{B} \sum_{i=1}^{B} \left( \frac{\partial L^{bf}(x_i; \theta)}{\partial \theta} |_{\theta^k} \right) _{\phi^k} \frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k}$$

$$= - \frac{\eta_\theta}{B} \sum_{j=1}^{B} \left( C^D(j), \frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k} \right) _{\phi^k} + \frac{1}{B} \sum_{i=1}^{B} \left( \frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k} \right) _{\phi^k},$$

(13)

where the last term in (13) is the derivative of the RRN $h_j(\phi)$ outputs of the $j$-th training sample pair with respect to the network parameters $\phi$, that is

$$\frac{\partial h(\phi)}{\partial \phi} = \left[ \frac{\partial \theta(\phi)}{\partial \phi} |_{\phi^k} \right] _{\phi^k}.$$
and the coefficients $G(j) = [G^D(j) \ G^M(j)]$, where

$$G^D(j) = \frac{1}{B} \sum_{i=1}^{B} \frac{\partial L^D_i(x_i; \theta)}{\partial \theta} \bigg|_{\theta^k} \frac{\partial L^r(x_j^D; \theta)}{\partial \theta} \bigg|_{\theta^k}$$

$$\leq \frac{1}{B} \sum_{i=1}^{B} g^{bf}(x_i) \cdot g^{tr}(x_j^D),$$

$$G^M(j) = \frac{1}{B} \sum_{i=1}^{B} \frac{\partial L^M_i(x_i; \theta)}{\partial \theta} \bigg|_{\theta^k} \frac{\partial L^r(x_j^M; \theta)}{\partial \theta} \bigg|_{\theta^k}$$

$$\leq \frac{1}{B} \sum_{i=1}^{B} g^{bf}(x_i) \cdot g^{tr}(x_j^M),$$

respectively. Denote the gradient of the meta loss of the $i$-th sample of $D^f$ as $g^{bf}(x_i) = \frac{\partial L^f(x_i; \theta)}{\partial \theta} \bigg|_{\theta^x}$, and the gradient of training loss on the $j$-th sample pair of $D^r$ as $g^{tr}(x_j^D) = \frac{\partial L^r(x_j^D; \theta)}{\partial \theta} \bigg|_{\theta^k}$ and $g^{tr}(x_j^M) = \frac{\partial L^r(x_j^M; \theta)}{\partial \theta} \bigg|_{\theta^k}$. Obviously, the coefficient $G(j)$ represents the similarity between the gradient of training loss $[g^{tr}(x_j^M) \ g^{tr}(x_j^f)]$ and the average of the gradient of meta loss $g^{bf}(x_j)$. Furthermore, we can reformulate the average gradient of the meta loss by class. The coefficient $G(j)$ can be represented as:

$$G^D(j) = \frac{1}{B} \sum_{c=1}^{C_i} \left( \frac{1}{B_c} \sum_{i=1}^{B_c} g^{bf}(x_i) \right) g^{tr}(x_j^D),$$

$$G^M(j) = \frac{1}{B} \sum_{c=1}^{C_i} \left( \frac{B_c}{B} \sum_{i=1}^{B_c} g^{bf}(x_i) \right) g^{tr}(x_j^M),$$

where $C_i$ is the number of all seen classes, $B_c$ is the sample number of each class in a batch, and $\sum_{c=1}^{C_i} B_c = B$. This formulation shows that the coefficients $G(j)$ implicitly model the relationship between the knowledge extracted from each training sample of the new task and that from the average meta samples. Then the first derivative term in (13) can be represented as:

$$\nabla_{\phi} \mathcal{L}^f(D^f; \phi) = -\frac{\eta_b}{B} \sum_{i=1}^{B} g^{bf}(x_i) \cdot \frac{\partial h_j(\phi)}{\partial \theta} \bigg|_{\theta^k}.$$  (17)

Therefore, the gradient of the RRN parameters $\phi$ can be calculated by (8), which can be easily done in PyTorch [53] with the automatic differentiation.

**APPENDIX B
MORE EXPERIMENT DETAILS**

In this section, we first illustrate some experimental details and then present some additional results to demonstrate the effectiveness of our method further.

**A. How to Apply Our Method to Other Baselines**

In the main text of the paper, we take ER as an example to illuminate how our proposed approach helps rehearsal-based continual learning models deal with the ‘stability-plasticity’ dilemma. Here we present how to apply our proposed method to other baselines, i.e., ER-ACE [29], and DER++ [26].
a) Relational-ER-ACE (RER-ACE): ER-ACE [29], which represents ‘Experience Replay with Asymmetric Cross-Entropy’, combines the losses of the new task samples and the memory buffer samples as

\[
L^{tr}(\theta) = \frac{1}{B} \sum_{i=1}^{B} \lambda^{D} L_{CE} (x^D_i; \theta, C_{curr}) \\
+ \lambda^{M} L_{CE} (x^M_i; \theta, C_{curr} \cup C_{curr}),
\]

(18)

where the hyperparameters \(\Lambda^D\) and \(\Lambda^M\) are preset to 1 in [29]. The \(L_{CE}(D; C)\) is defined as:

\[
L_{CE}(x; \theta, C) = -\log \frac{z_c(x)}{\sum_{c \in C} z_c(x)},
\]

(19)

where the sample \(x\) belongs to the \(c\)-th class and \(z_c(x)\) is the \(c\)-th element of the main classification network output \(f(x; \theta)\).

Obviously, it is straightforward to apply our proposed method to ER-ACE, where the RRN still generates the weights \([\Lambda^D, \Lambda^M]\) for each training sample pair. The outer-loop optimization is the same as (4) and the generated sample weights are applied to the inner-loop optimization (18).

b) Relational-DER (RDER): DER++ introduce an additional knowledge distillation function \(L_{KD}(x^M_i; \theta)\) to minimize the distance between predict logits and saved previous logits, which is controlled by the weight \(\lambda^{KD}\) based on (1). Intuitively, the outer-loop loss function for the RRN in (4) can be reformulated as:

\[
L^{bf}(x; \theta) = L_{CE}^{bf}(x; \theta) + L_{KD}^{bf}(x; \theta),
\]

(20)

which is the sum of the CE loss and the distillation loss.

B. More Details for Hyperparameters Setting

In the warm-up stage (i.e. the epochs before \(Iter_{warm}\)), we use preset weights in the inner loop optimization like previous methods. Specifically, for RER and RER-ACE, the weight for the CE loss of new and old task samples is preset as 1 and 0.5, respectively. And for RDER, the preset weights are 1, 0.5, and 0.2 for the CE loss of new task samples \(L_{CE}(D_t)\), the CE loss of memory buffer samples \(L_{CE}(M_t)\), and the distillation loss of memory buffer samples \(L_{KD}(M_t)\), respectively.

C. Additional Visualization of Each Task Accuracy

Here we show the classification accuracy of each task of ER-ACE/RER-ACE and DER++/RDER on CIFAR-100 in Fig. 7(a) and (c), and on Tiny ImageNet in Fig. 7(b) and (d). Similar to Fig. A1, our method obviously improves the accuracy of the previous tasks. Besides, to balance ‘stability’ and ‘plasticity’, RDER achieves an overall higher performance, even though it may not outperform DER++ on some new tasks as shown in Fig. 7(c). All of these results further demonstrate the effectiveness of our method, which can easily be adapted to multiple rehearsal-based methods.

D. Additional Visualization of Weight Distribution

Here we show more weight distribution of different stages in the model training. As shown in Fig. 8, in different stages of training, weight distributions are dynamically changing, which also demonstrates that the sample weight should not be prefixed. In addition, buffer sample weights are usually larger at the beginning of training. This is because the model has not yet adapted to new samples, resulting in a greater impact on old knowledge, so larger weights may be needed to avoid forgetting. As training proceeds, the impact of task correlation on weight distribution gradually appears, reflecting the rules we analyzed previously in Section V-D.

Furthermore, to explore how RRN generates weights for unseen data samples, we also display the average weight distribution on the test dataset under Setup 1 and Setup 2. We use the test samples of task 2 and task 1 to construct the paired data as the inputs of RRN, and then we calculate the average of all generated weights. To avoid stochastic factors, we repeat our experiment 10 times and show their average weights in Fig. 9. We can observe that the average weights of Setup 1 mainly concentrate on the left upper corner of Fig. 9. That is, for similar tasks, buffer sample weights are relatively larger than new weights to prevent forgetting, the same as the trend of training sample weights in Fig. 8. On the other hand, the average weights of Setup 2 are relatively separated. These results further demonstrate that our RRN has extracted task relationships for generating weights and also works on unseen test data, rather than simply overfitting on the training data.

E. Computation Cost and Acceleration Strategy

The proposed method involves a bi-level optimization that introduces an additional calculation burden primarily due to the second-order derivation in the outer loop optimization. To mitigate this, we employ Interval to regulate the update frequency of RRN to strike a better trade-off between the model performance and computation cost, which is a common way to speed up the bi-level optimization [47]. As shown in Table X, the memory usage of RER with Interval = 5 is approximately 500 MB higher than that of ER, and the training time of RER is roughly twice that of ER. Importantly, it’s worth noting that the proposed RRN and bi-level optimization procedures are only involved in the training stage, meaning that no computational overhead is required during testing.

Fortunately, many previous works are trying to accelerate this bi-level optimization framework, such as some first-order approximation [57]. We find that reducing the parameters involved in the second-order derivation of the outer loop optimization can greatly enhance the efficiency of bi-level optimization, with little impact on performance [58], [59], [60]. To speed up the optimization of RRN, we assume that the parameter \(\phi\) of RRN only correlates to the final linear classification layer of the main net, rather than the entire network. With this assumption, the calculating of second-order derivation only involves a few parameters, which can significantly accelerate the optimization of our bi-level framework. For example, if we use ResNet-18 as the main net for the CIFAR-10 dataset, the final classifier comprises merely 5,120 parameters, whereas the entire network contains about 11.69 M parameters. So the second-order derivation will be faster and more computation-friendly. As shown in Table X,
Fig. 8. Weight distributions at multiple epochs during training. The first two columns are weight distributions for buffer samples and new task samples in Setup 1, while the last two columns are those for buffer samples and new task samples in Setup 2. Each row represents the weight distribution of epochs 5, 15, 25, and 35 of the whole training process of task 2, and the weight distributions of epoch 45 are shown in Fig. 6.

**TABLE X**

**COMPUTATION RESOURCE BASED ON NVIDIA GeForce RTX 2080Ti**

| Method  | Memory (Mb) | Average Training Time per Epoch (s) | Average Training Time per Iteration (s) | ACC on CIFAR-10 with M=200 |
|---------|-------------|-------------------------------------|----------------------------------------|-----------------------------|
| ER      | 1584        | 21.82                               | 0.0547                                 | 55.84                       |
| RER     | 2080        | 40.95                               | 0.1156                                 | 58.59                       |
| FastRER | 1712        | 29.05                               | 0.0805                                 | 58.18                       |
we denote this accelerated method FastRER, which only increases memory usage by about 200MB and training time by about 7.2 seconds. Furthermore, FastRER achieves comparable performance to RER.

Fig. 9. Scatter plot of the average sample weights under Setup 1 and Setup 2. The horizontal and vertical axes represent the sample weight magnitude of the new and old tasks, respectively.

REFERENCES

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Represent., 2015, pp. 1–14.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[3] F. Iandola et al., “DenseNet: Implementing efficient ConvNet descriptor pyramids,” 2014, arXiv:1404.1869.

[4] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 2881–2890.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 2961–2969.

[6] A. Douillard, Y. Chen, A. Dupont, and M. Cord, “PLOP: Learning without forgetting for continual semantic segmentation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 4040–4050.

[7] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. Int. Conf. Adv. Neural Inf. Process. Syst., 2015, vol. 28, pp. 91–99.

[8] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” 2018, arXiv:1804.02767.

[9] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” 2020, arXiv:2004.10934.

[10] M. McCloskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” in Psychology of Learning and Motivation, vol. 24, Amsterdam, The Netherlands: Elsevier, 1989, pp. 109–165.

[11] M. De Lange et al., “A continual learning survey: Defying forgetting in classification tasks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 7, pp. 3366–3385, Jul. 2022.

[12] A. A. Rusu et al., “Progressive neural networks,” 2016, arXiv:1606.04671.

[13] C. Fernando et al., “PathNet: Evolution channels gradient descent in super neural networks,” 2017, arXiv:1701.08734.

[14] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, “Overcoming catastrophic forgetting with hard attention to the task,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 4548–4557.

[15] D. Abati et al., “Conditional channel gated networks for task-aware continual learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 3931–3940.

[16] S. Yan, J. Xie, and X. He, “DER: Dynamically expandable representation for class incremental learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 3014–3023.

[17] J. Kirkpatrick et al., “Overcoming catastrophic forgetting in neural networks,” Nat. Acad. Sci., vol. 114, no. 13, pp. 3521–3526, 2017.

[18] Z. Li and D. Hoiem, “Learning without forgetting,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 12, pp. 2935–2947, Dec. 2018.

[19] T. Zenke, B. Poole, and S. Ganguli, “Continual learning through synaptic intelligence,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 3987–3995.

[20] J. Schwartz et al., “Progress & compress: A scalable framework for continual learning,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 4528–4537.

[21] H. Yin, P. Yang, and P. Li, “Mitigating forgetting in online continual learning with neuron calibration,” in Proc. Int. Conf. Adv. Neural Inf. Process. Syst., 2021, vol. 34, pp. 10260–10272.

[22] R. Ratcliff, “Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions,” Psychol. Rev., vol. 97, no. 2, pp. 285–308, 1990.

[23] A. Robins, “Catastrophic forgetting, rehearsal and pseudo rehearsal,” Connection Sci., vol. 7, no. 2, pp. 133–146, 1995.

[24] S. A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “iCaRL: Incremental classifier and representation learning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 2001–2010.

[25] D. Lopez-Paz and M. Ranzato, “Gradient episodic memory for continual learning,” in Proc. Int. Conf. Adv. Neural Inf. Process. Syst., 2017, pp. 6467–6476.

[26] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, “Dark experience for general continual learning: A strong, simple baseline,” in Proc. Int. Conf. Adv. Neural Inf. Process. Syst., 2020, vol. 33, pp. 15920–15930.

[27] A. Chaudhry, A. Gordo, P. Dokania, P. Torr, and D. Lopez-Paz, “Using hindsight to anchor past knowledge in continual learning,” in Proc. AAAI Conf. Artif. Intell., 2021, vol. 35, pp. 6993–7001.

[28] J. Bang, H. Kim, Y. Yoo, J.-W. Ha, and I. Choi, “Rainbow memory: Continual learning with a memory of diverse samples,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2021, pp. 8218–8227.

[29] L. Caccia et al., “New insights on reducing abrupt representation change in online continual learning,” in Proc. Int. Conf. Learn. Representations, 2022, pp. 1–27.

[30] H. Ahn et al., “SS-IL: Separated softmax for incremental learning,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2021, pp. 844–853.

[31] G. A. Carpenter and S. Grossberg, “A massively parallel architecture for a self-organizing neural pattern recognition machine,” Comput. Vis., Graph., Image Process., vol. 37, no. 1, pp. 54–115, 1987.

[32] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, “BBN: Bilateral-branch network with cumulative learning for long-tailed visual recognition,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 9716–9725.

[33] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin, “Learning a unified classifier incrementally via rebalancing,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 831–839.

[34] A. Chaudhry, R. Marc’Aurelio, M. Rohrbach, and M. Elhoseiny, “Efficient lifelong learning with A-GEM,” in Proc. Int. Conf. Learn. Representations, 2019, pp. 1–20.

[35] A. Mallya and S. Lazebnik, “PackNet: Adding multiple tasks to a single network by iterative pruning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 7765–7773.

[36] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, “Variational continual learning,” in Proc. Int. Conf. Learn. Representations, 2018, pp. 1–18.

[37] A. Maracani, U. Michieli, M. Toldo, and P. Zanuttigh, “RECALL: Replay-based continual learning in semantic segmentation,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2021, pp. 7026–7035.

[38] C.-B. Zhang, J.-W. Xiao, X. Liu, Y.-C. Chen, and M.-M. Cheng, “Representation compensation networks for continual semantic segmentation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2022, pp. 7053–7064.

[39] G. Yang et al., “Continual attentive fusion for incremental learning in semantic segmentation,” IEEE Trans. Multimedia, vol. 25, pp. 3841–3854, 2023.

[40] Y. Tao et al., “Few-shot class-incremental learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 12183–12192.

[41] D.-W. Zhou et al., “Forward compatible few-shot class-incremental learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2022, pp. 9046–9056.

[42] Y. Cui et al., “Uncertainty-guided semi-supervised few-shot class-incremental learning with knowledge distillation,” IEEE Trans. Multimed., vol. 25, pp. 6422–6435, 2023.

[43] S. Thuseethan, S. Rajasegarar, and J. Yearwood, “Deep continual learning for emerging emotion recognition,” IEEE Trans. Multimed., vol. 24, pp. 4367–4380, 2022.
Yuexiang Li received the Ph.D. degree from the University of Nottingham, Nottingham, U.K. He is a Full Professor with the Guangxi Key Laboratory for Genetic and Personalized Medicine, Guangxi Medical University, Nanning, China. He is the academic leader of the discipline of artificial intelligence, and leads the Medical AI Research group in the university. His research interests include intelligent analysis and processing of medical images including microscopic images, pathological slices, and multimodal medical images.

Dong Wei received the Ph.D. degree in computer engineering from the University of Singapore, Singapore, in 2013. Since 2018, he has been a Senior Researcher with Tencent Jarvis Lab, Shenzhen, China. His research interests include medical image analysis, with a current focus on data and annotation efficient approaches.

Hong Wang received the Ph.D. degree from the School of Mathematics and Statistics, Xi’an Jiaotong University, Xi’an, China, in 2021, under the supervision of Prof. Deyu Meng. From 2020 to 2021, she was a research intern with Tencent Rhino Bird Elite Talent Program. She is currently a Senior Researcher with Tencent, Shenzhen, China. Her research interests include the design of model-driven and data-driven deep learning techniques for effective and interpretable image processing including natural image restoration and medical image analysis.

Kai Ma received the Ph.D. degree from the University of Illinois at Chicago, Chicago, IL, USA, in 2014. He was with Siemens Medical Solution, Princeton, NJ, USA, for more than five years. He is currently a Principal Researcher with Tencent Jarvis Lab, Shenzhen, China. His research interests include medical image analysis, deep learning, computer vision, and brain-computer interface.

Yefeng Zheng (Fellow, IEEE) received the B.E. and M.E. degrees from Tsinghua University, Beijing, China, in 1998 and 2001, respectively, and the Ph.D. degree from the University of Maryland, College Park, MD, USA, in 2005. After graduation, he joined Siemens Corporate Research, Princeton, NJ, USA. He is currently the Director and Distinguished Scientist of Tencent Jarvis Lab, Shenzhen, China, leading the company’s initiative on Medical AI. His research interests include medical image analysis, computer vision, natural language processing, and deep learning. Dr. Zheng is a Fellow of the American Institute for Medical and Biological Engineering.

Deyu Meng (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Xi’an Jiaotong University, Xi’an, China, in 2001, 2004, and 2008, respectively. From 2012 to 2014, he was a Visiting Scholar with Carnegie Mellon University, Pittsburgh, PA, USA. He is currently a Professor with the School of Mathematics and Statistics, Xi’an Jiaotong University, and adjunct Professor with the Faculty of Information Technology, Macau University of Science and Technology, Macau, China. His research interests include model-based deep learning, variational networks, and meta-learning.