Rebuttal Response

Anonymous Authors[1]

1. General Response

- do not cover all CL methods, add ER, fails on counterexample, and change statement to be weaker
- CL formulation, solving sub-problem correctly offers anytime guarantee, it implies the averaged learning and forgetting guarantee, Theorem 4.1 (Lin et al., 2023)
- add class-incremental, theory can cover
- already have multiple runs, add figures
- ablation study on hyperparameter
- recent baselines

2. Reviewer 2wx4 (rating 4 confidence 5)

Q1. The motivation or the presentations need to be revised. The paper and the proposed method are mainly targeting the continual learning (CL) problem formulated as Eq. (1) or (5). It is not the original objective of continual learning. In general (memory-based) CL problem, solving the formulated optimization cannot guarantee any good results. Firstly, it is just a subproblem with a given set of memory, at a static step in CL. How to maintain the memory also influences the results. On the other hand, even considering the problem with a given memory, it is questionable that solving the formulated constrained optimization problem is the solution, as only one specific way to solve the learning problem at a single step. In principle, solving this constrained problem can be a design, but has specific limits, such as avoiding potential knowledge sharing/back-transfer. The theory and proof are also just for the given sub optimization problem, instead of the whole CL problem. The authors may consider revising the presentations (especially in the introduction and title) to avoid potentially misleading information.

A1. Thanks for your insightful question. Note that our focus here is to enable the continual learner to learn the current task without forgetting the previously learned knowledge *at any time*. It is an important property, especially for online learning and online continual learning (Cutkosky, 2019; Caccia et al., 2022), where people care about any time accuracy. Note that the if the following formulation (1) holds for every task t, then we can guarantee that there is no forgetting at any time, and it implies that there is no forgetting in the averaged sense as well. We will include this paragraph in the revised version of the paper.

Now let us explain how it works. The formulation (1) is defined as the following:

$$\begin{aligned} &\min_{\theta_t} \ell(\theta_t, \mathcal{D}_t^{tr}) \\ &\text{s.t., } \ell(\theta_t, \mathcal{M}_i) \le \ell(\theta_{t-1}, \mathcal{M}_i),\ \forall i < t, \end{aligned} \tag{1}$$

where θ_t is the model parameter of the current task t, θ_{t-1} is the model parameter trained after previous task $t-1$ and $\mathcal{M}_i$ is the memory buffer for task i. Due to the limitation access to the previous tasks, CL model can only maintain a small buffer

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

to store the history data. Since we solve the formulation (1) for every task t, we can have anytime learning and forgetting guarantees.

Now we show that it implies the average forgetting guarantees if we solve (1). The averaged forgetting measure of the model after training over all T tasks can be defines as (Chaudhry et al., 2018; Lin et al., 2023):

$$F_T = \frac{1}{T-1}\sum_{t=1}^{T-1}(\ell(\theta_T, \mathcal{M}_t) - \ell(\theta_t, \mathcal{M}_t)). \tag{2}$$

In fact, solving (1) for each task t can guarantee $F_T \leq 0$ in (2), since for any $t \in [T-1]$, we have

$$\begin{aligned}\ell(\theta_T, \mathcal{M}_t) - \ell(\theta_t, \mathcal{M}_t) &= \ell(\theta_T, \mathcal{M}_t) - \ell(\theta_{T-1}, \mathcal{M}_t)\\ &\quad + \ell(\theta_{T-1}, \mathcal{M}_t) - \ell(\theta_{T-2}, \mathcal{M}_t) + \cdots + \ell(\theta_{t+1}, \mathcal{M}_t) - \ell(\theta_t, \mathcal{M}_t) \leq 0,\end{aligned}$$

due to the constraint of (1). Then it implies $F_T \leq 0$. That is, we can indeed avoid the catastrophic forgetting for the whole CL problem in terms of definition (2).

Q2. In Sec 5.1, the authors listed the challenges and issues in previous methods. But it is unclear how the proposed method can address it in principle. For example, I cannot find theoretical or at least in-detailed discussions to support the benefits on batch size of proposed, in the paragraph related to this discussion. I can see some claims. But more detailed justification or analysis is needed, as a main claim.

A2. As mentioned in Section 5.1, existing algorithms which can solve the original constrained optimization (eq (1) in the submitted paper) in our problem setting rely on large batch size (that is, batch size depend on $1/\epsilon$, e.g., Theorem A.14 in (Huang & Lin, 2023)) or double-loop design (e.g., Theorem 3 in (Ma et al., 2020)), which is not practical for CL because practical CL algorithms are usually single-loop and only use small batch size (here "small batch size" refers to independence of $1/\epsilon$ theoretically) to learn from continuous data streams.

To this end, we introduce the momentum-style moving-average estimator (line 8 in Algorithm 1) to approximate the forgetting measure at every iteration. This approach utilizes history information and avoids using large mini-batch. As can be seen from Theorem B.10 in Appendix B, our theoretical result does not pose any requirement on batch size, as opposed to (Huang & Lin, 2023). In other words, our proposed algorithm works for any batch size.

Q3. The experiments are limited.

- What is the setting used in experiments (in Fig 3), class incremental learning or task-incremental learning?

  A. Task-incremental learning is used in the original paper. Actually, we can extend our algorithm to class-incremental scenario seamlessly. Instead of using customized classifiers related to the different task in the last NN layer, a single classifier is shared across all CL tasks. We conducted class-incremental experiments on Multiple Dataset and Split CIFAR-10 with noise-free ($p = 0.0$) and label noise ($p = 0.2$) settings. Multiple Dataset consists of five different image datasets and the model will encounter a new dataset in each subsequent task. CIFAR-10 is splitted into 5 sequential tasks, each of which contains two disjoint classes. We report the test accuracy with 5 independent runs on Multiple Dataset and Split CIFAR-10 in Table 1 and Table 2. For fair comparison, we run all algorithms for 100 epochs on Multiple Dataset with step size 0.05 and memory size 256, 50 epochs on Split CIFAR-10 with step size 0.03 and memory size 512. For STREAM, the forgetting threshold is set to $\gamma = 0.005$ in Multiple Dataset, and $\gamma = 0.05$ in Split CIFAR-10.

  As can be seen from Table 1 and 2, SLIP surpasses all the baselines in terms of test accuracy. It achieves 4.93%, 5.97% higher average accuracy than the second best ones on noise-free ($p = 0.0$) and noise ($p = 0.2$) setting of Multiple Dataset, and 6.60%, 5.35% higher average accuracy on Split CIFAR-10. Besides, SLIP exhibits competitive forgetting measure compared with other baselines.

- Ablation studies of the main components in the algorithm are needed. The main algorithm contains several components, such as the moving average estimator and switching branch. The authors should conduct experiments by removing either of these components (to study the importance) as well as removing all of them (to prove fair comparison with the other methods).

Table 1. Results of class-incremental experiments on Multiple Dataset (p denotes the noise rate).

Methods	$p = 0.0$		$p = 0.2$	
	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
EWC	13.08±5.36	0.695±0.004	10.36±3.21	0.590±0.009
MAS	10.06±1.13	0.613±0.019	6.04±0.62	0.487±0.016
AGEM	13.67±1.24	0.654±0.018	12.10±1.24	0.548±0.007
OGD	13.00±0.74	0.707±0.005	9.75±0.44	0.586±0.005
DER	45.67±2.19	0.260±0.039	40.88±2.15	0.248±0.026
GDumb	40.69±3.32	**0.128±0.023**	36.25±2.23	**0.083±0.019**
MEGA	38.94±2.79	0.389±0.026	34.03±2.23	0.134±0.023
STREAM	**47.92±0.45**	0.153±0.018	**43.32±0.018**	0.367±0.013

Table 2. Results of class-incremental experiments on Split CIFAR-10 (p denotes the noise rate).

Methods	$p = 0.0$		$p = 0.2$	
	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
EWC	18.36±0.23	0.714±0.018	18.01±0.23	0.717±0.007
MAS	14.36±1.13	**0.275±0.021**	11.66±1.59	0.546±0.020
AGEM	20.50±1.92	0.693±0.035	19.44±1.14	0.525±0.071
OGD	19.02±0.12	0.734±0.003	18.99±0.08	0.736±0.004
DER	29.87±2.52	0.586±0.084	22.62±3.27	0.685±0.029
GDumb	28.40±2.13	0.365±0.054	21.69±0.59	**0.227±0.010**
MEGA	33.16±3.12	0.471±0.106	24.84±2.26	0.475±0.091
STREAM	**35.35±3.52**	0.504±0.098	**26.17±1.32**	0.303±0.056

A. We have conducted ablation studies about (i) moving averaging estimator in section 6.5 of the submitted paper. (ii) switching gradient update rule in Appendix A.4.2 of the submitted paper. (i) To study the effect of moving averaging estimator, we conduct an ablation study to compare the model performance by setting $\beta = 0.0$ and $\beta = 0.9$ in our algorithm, where $\beta = 0.0$ indicates the moving-average estimator is removed and we only use one minibatch to estimate the forgetting measure. We compared the model with ($\beta = 0.9$) and without the moving-average estimator ($\beta = 0.0$). The results are shown in Figure 4 (in original submission), which shows that the algorithm with the moving-average estimator performs better throughout the whole learning process. (ii) The forgetting threshold γ controls the swiching gradient branch. Large thresholds would render the model update more on the current data, otherwise update more on the memory data. To study the impact of γ to model performance, we run STREAM algorithm with different values of γ, you can find the results in Figure 9 (in Appendix A.4.2 of the original submission). The empirical results are consistent with our analysis. Too large (≥ 1.0) value of γ actually deactivates the switching gradient update. Instead, an appropriately small γ works well for all experiments. For your question"removing all of the": If we do like that, that destroys the original framework of our algorithm, our algorithm actually cannot be executed.

- The running time should be analyzed and reported.

A3. To assess the running time of various algorithms, we execute them independently on the device (NVIDIA A6000 48GB GPU). The elapsed wall-clock time was recorded throughout the entire training process, which encompassed 25 epochs on the standard (without noise) Multiple Dataset and 10 epochs on Split CIFAR-100. The evaluation results are summarized in Table 3. Notably, STREAM emerges as the algorithm with the fastest training speed. Unlike other algorithms using the mixed gradient (current gradients and memory gradients) to update model parameters for each update iteration, STREAM adopts switching gradient which would chooses either one of the gradients (current gradients or memory gradients) to implement the backpropagation in each iteration, which makes STREAM significantly more time-efficient.

3. Reviewer AHWH (rating 6 confidence 4)

Q1. (Minor point) From what I understand, this is task-incremental learning. If so, this should be mentioned early on (eg in the Intro), along with a discussion of how easily or well this might extend to more general CL (eg class-incremental or

Table 3. Running time on Multiple Dataset and Split CIFAR-100.

Methods	Multiple Dataset (hours)	Split CIFAR-100 (hours)
EWC	0.16	1.31
MAS	0.17	1.31
AGEM	0.16	1.30
OGD	0.47	3.01
DER	0.18	1.27
GDumb	0.13	0.92
MEGA	0.15	1.05
STREAM	**0.11**	**0.79**

beyond). Many recent algorithms in memory-based CL tackle at least class-incremental learning.

A1. Thanks for your insightful reminder. As you mentioned, our experimental results only include task-incremental learning in the original submitted paper. However, we want to emphasize that our algorithmic design can handle class-incremental scenario seamlessly. Instead of using customized classifiers for different tasks in the last NN layer, a single classifier is shared across all CL tasks.

Let us first explain why our algorithm can handle class-incremental continual learning from a theoretical perspective: it actually aligns more closely with our theoretical framework. In Algorithm 1 of submitted paper, we presumably need to update θ, which contains all the model parameters of a neural network. Considering the task-incremental scheme, θ includes the parameter of the previous layers θ^{rep} to learn the representation and the parameter of the last linear layer as a classifier $\theta^{cl} = (\theta_1^{cl}, \dots, \theta_T^{cl})$, where θ_i^{cl} stands for the linear layer for i-th task and $1 \leq i \leq T$. However, in practice, we do not necessarily need to update all the parameters. In practical CL (Chaudhry et al., 2018), when learning t-th task, only θ_t^{cl} and θ^{rep} need to be updated and the rest parameters are frozen. We show that our algorithm and theory are also applicable in this case. At the beginning of the t-th task, we need to update set $\mathcal{K} = \mathcal{K} \cap \mathcal{K}_t$ in Algorithm 1, where $\mathcal{K}_t = \{(\theta^{rep}, \theta_1^{cl}, \dots, \theta_T^{cl}) \mid \theta_i^{cl} = \theta_i^{cl'}, \forall i \neq t\}$ and $\theta_i^{cl'}$ denotes the i-th task's linear layer at the end of $(t-1)$-th task. If we consider class-incremental case, we just update all the model parameter θ, which is continuously changing across different sequential tasks, and therefore we do not need to change the projection set for different iterations. In addition, the assumptions needed for theoretical proofs are automatically satisfied. Our current theory actually covers the class-incremental case, and even exhibits higher consistency with class-incremental continual learning.

Empirically, we conducted class-incremental experiments on Multiple Dataset and Split CIFAR-10 with noise-free ($p = 0.0$) and label noise ($p = 0.2$) settings. Multiple Dataset consists of five different image datasets and the model will encounter a new dataset in a each subsequent task. CIFAR-10 is divided into 5 sequential tasks, each of which contains two disjoint classes. We report the test accuracy with 5 independent runs on Multiple Dataset and Split CIFAR-10 in Table 1 and Table 2. For fair comparison, all the algorithms run for 100 epochs on Multiple Dataset with step size 0.05 and memory size 256, 50 epochs on Split CIFAR-10 with step size 0.03 and memory size 512. For STREAM, the forgetting threshold is set to $\gamma = 0.005$ in Multiple Dataset, and $\gamma = 0.05$ in Split CIFAR-10.

As can be seen from Table 1 and 2, SLIP surpasses all the baselines in terms of test accuracy. It achieves 4.93%, 5.97% higher average accuracy than the second best ones on noise-free ($p = 0.0$) and noise ($p = 0.2$) setting of Multiple Dataset, and 6.60%, 5.35% higher average accuracy on Split CIFAR-10. Besides, SLIP exhibits competitive forgetting measure compared with other baselines.

Q2. Although I really liked the counterexample, it strikes me that this issue will likely not arise in neural networks because NNs have large capacity and they can likely change other, less-important, parameters instead of $\theta^{(1)}$. Do the authors agree? Regardless, I think there should be a couple of lines of discussion as to why GEM, A-GEM and OGD work decently in practice, and that this counterexample serves a purpose to show some limitations that, although exaggerated in this example, may lead to (smaller) performance decreases in practice.

A2. Our counterexample is not a neural network, but whether it would occur in neural networks is an open problem. We agree that this issue will likely not arise in neural networks for the current benchmark datasets, but it may fail for other task data distributions. We plan to investigate such cases in future work.

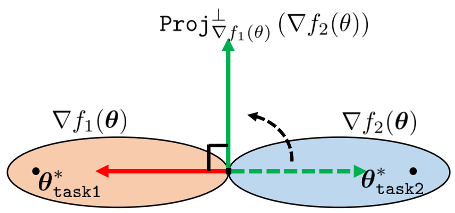

$\mathtt{Proj}^{\perp}_{\nabla f_1(\theta)}(\nabla f_2(\theta))$
$\nabla f_1(\boldsymbol{\theta})$
$\nabla f_2(\boldsymbol{\theta})$
$\boldsymbol{\theta}^*_{\mathtt{task1}}$
$\boldsymbol{\theta}^*_{\mathtt{task2}}$

Figure 1. The Failure example for GEM, A-GEM and OGD, where the current gradient $\nabla f_2(\theta)$ is in the opposite direction to the memory gradient $\nabla f_1(\theta)$. The update direction is based on a rotation of the current gradient, which is orthogonal to the memory gradient: $\mathtt{Proj}^{\perp}_{\nabla f_1(\theta)}(\nabla f_2(\theta))$. Therefore it cannot learn task 2.

The intuition for constructing this counterexample is that there exists some cases where the data distribution of new tasks is significantly different from the old tasks such that the current gradient direction can be just opposite to the memory gradient direction. As a result, A-GEM and OGD will project the current gradient to be orthogonal to the memory gradient or base, which fails to learn the new task, as illustrated in Figure 1. Such case exists commonly in reality, e.g., a pre-trained foundation model is continually adapted to a brand new domain. Multiple Dataset is somehow a demonstration of this case, where there are 5 different image datasets and each one is used to constructed a task. As you can see in Table 2 (Appendix A.2) of the submitted paper, A-GEM (33.47 ± 1.61) and OGD (27.88 ± 1.23) perform much worse than STREAM (72.08 ± 1.40) when $p = 0.0$, the huge performance gaps can be observed in the setting of higher noise level as well. Thus, noise is also another factor which interferes the gradient update of A-GEM and OGD. In other words, the noise can increase the possibility of the case occurring as shown in Figure 1, leading to lower accuracy than expected.

Q3. (Important point) The results need to be run with many random seeds and the plots and results need to show means and standard deviations. I cannot tell if any of the results are currently significant or not. In fact, I expect that some of the results are not significant given the small increases in accuracy (but happy to be shown wrong).

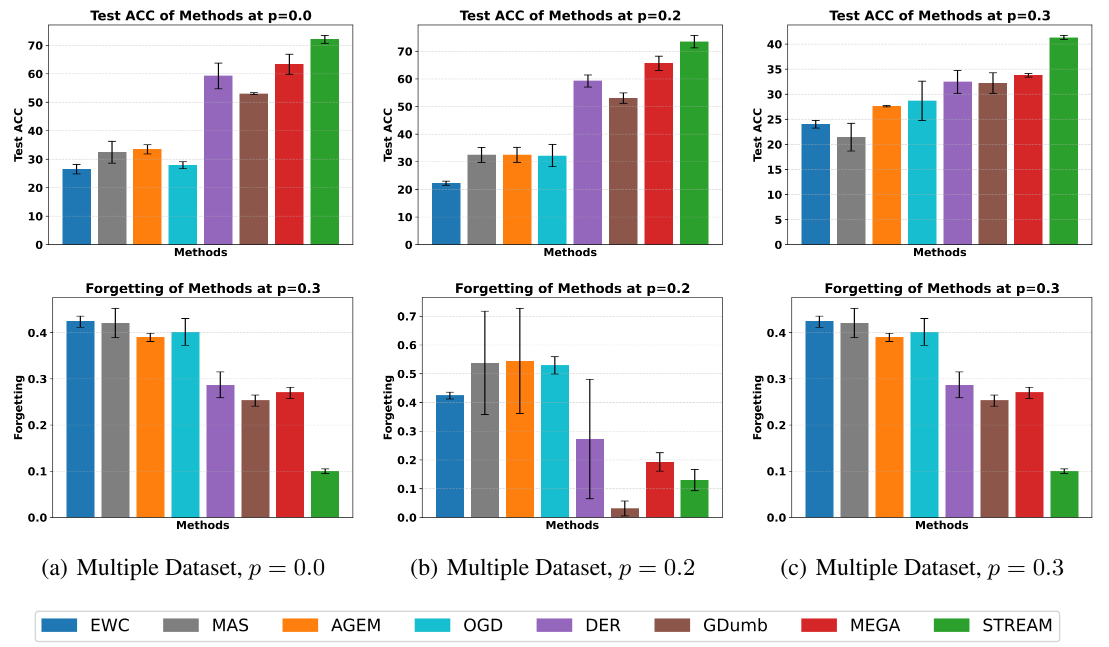
A3. Thanks for your good suggestion. Our experiments were executed in 5 independent runs with random seeds $\{0, 1, 2, 3, 4\}$, and we follow A-GEM (Chaudhry et al., 2018) (Figure 1 in their paper), OGD (Farajtabar et al., 2019) (Figure 3, 4 in their paper) and MEGA (Guo et al., 2020) (Figure 1 in their paper) to report the average test accuracy (with standard deviation) and forgetting (with standard deviation) of all the algorithms. The results corresponding to Multiple Dataset, Split CIFAR-100, Split Tiny-Imagenet, AVE-CI are shown in Figure 2, 3, 4, 5 respectively. There are two rows in each figure, where the first row shows the test accuracy and the second row shows the forgetting. STREAM consistently outperforms the other baselines in accuracy and enjoys a low forgetting.

Q4. (Important point) I was surprised by the low performance of all algorithms on some of the task-incremental (single modality) benchmarks. For example, looking at the DER paper on Split Tiny-ImageNet, DER obtained 40.87% accuracy with buffer size 200 (see Table 2 in their paper), while in this paper, DER obtained 18% under $p = 0$ with buffer size 256 (and STREAM achieves 31.36%). Why is there such large differences in performance?

A4. The performance difference mainly stems from two aspects. (1) The number of tasks. We construct 20 tasks with 10 classes in each task rather than 10 tasks considered in DER paper, and more tasks causes more severe forgetting. (2) The number of epochs. There are only 15 training epochs for each task in our experiment, in contrast to 100 epochs in DER. We believe fewer training epochs are preferred in the field of continual learning. For example, only one epoch is allowed in online continual learning.

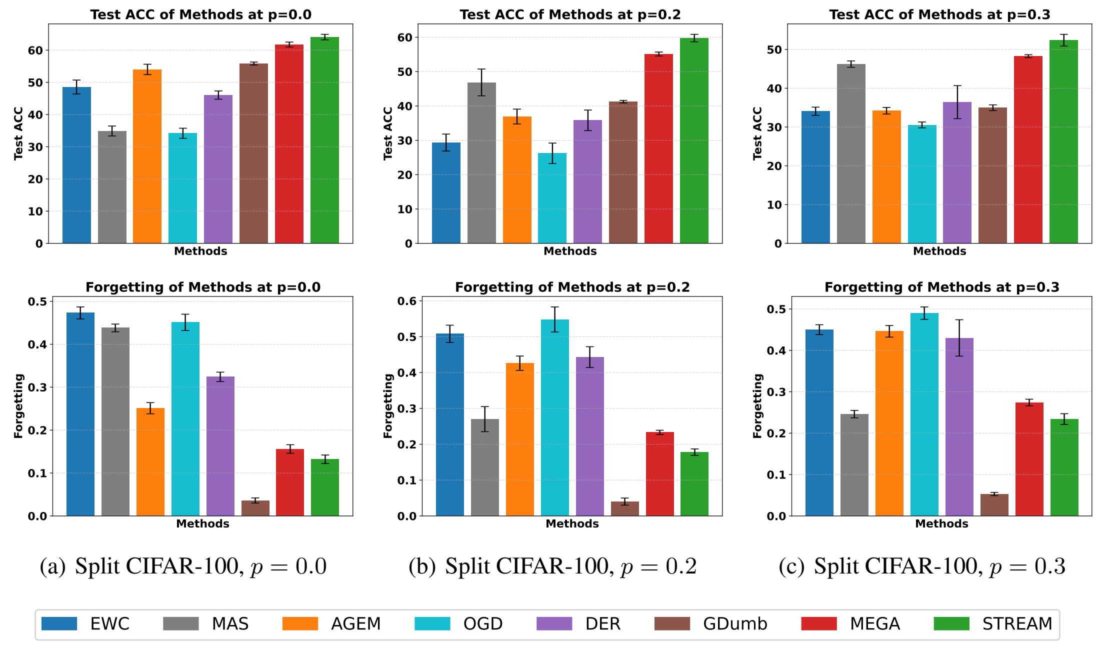
Q5. Typos / more minor suggestions.

A5. Thank you for your suggestions. We will fix the typos and add comments to highlight the key lines in Algorithm 1 to make it easier to follow in the revised version.



(a) Multiple Dataset, $p = 0.0$ (b) Multiple Dataset, $p = 0.2$ (c) Multiple Dataset, $p = 0.3$

Figure 2. Performance (test accuracy and forgetting) of continual learning methods on Multiple Dataset over 5 runs.



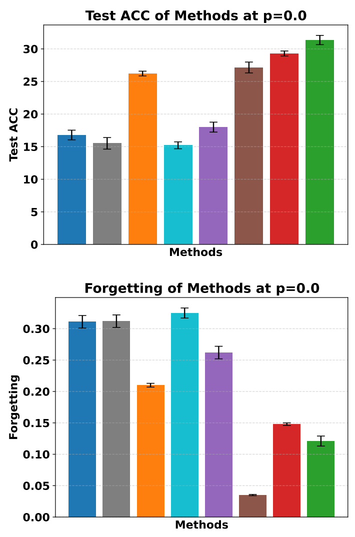
(a) Split CIFAR-100, $p = 0.0$ (b) Split CIFAR-100, $p = 0.2$ (c) Split CIFAR-100, $p = 0.3$

Figure 3. Performance (test accuracy and forgetting) of continual learning methods on Split CIFAR100 over 5 runs.

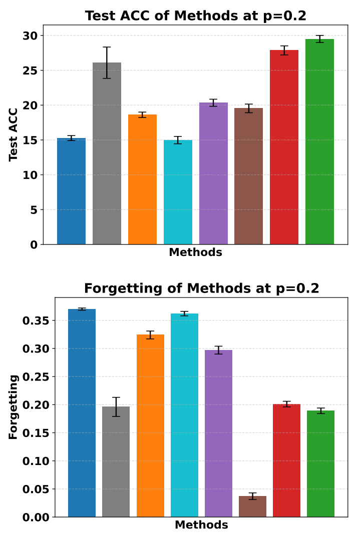
4. Reviewer F1s9 (rating 3 confidence 4)

Q1. The motivation and experiments shown in Section 4 have a big flaw. The primary motivation of the authors is: Do existing memory-based CL methods enjoy provable learning and forgetting guarantees for continual learning problems?

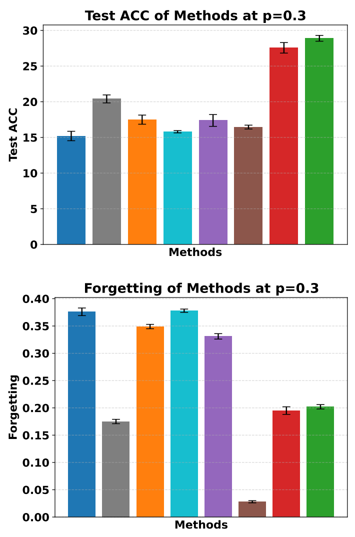
- However, the 3 methods selected as "representative" memory-based methods only follow one very specific and not very useful way to use the memory. GEM, AGEM and OGD are regularization methods that use a buffer but are



(a) Split Tiny-Imagenet, $p = 0.0$



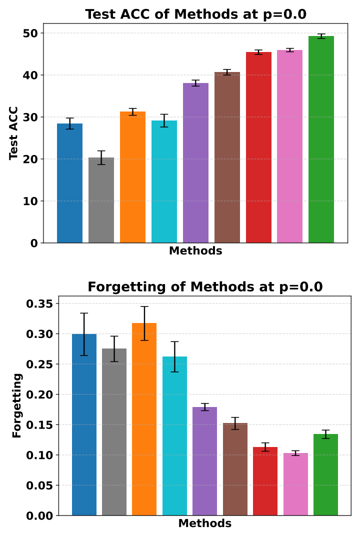
(b) Split Tiny-Imagenet, $p = 0.2$



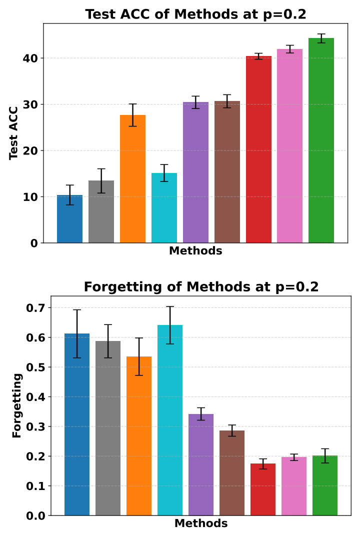
(c) Split Tiny-Imagenet, $p = 0.3$

🟦 EWC ⬜ MAS 🟧 AGEM 🟦 OGD 🟪 DER 🟫 GDumb 🟥 MEGA 🟩 STREAM

Figure 4. Performance (test accuracy and forgetting) of continual learning methods on Split Tiny-Imagenet over 5 runs.



(a) AVE-CI, $p = 0.0$



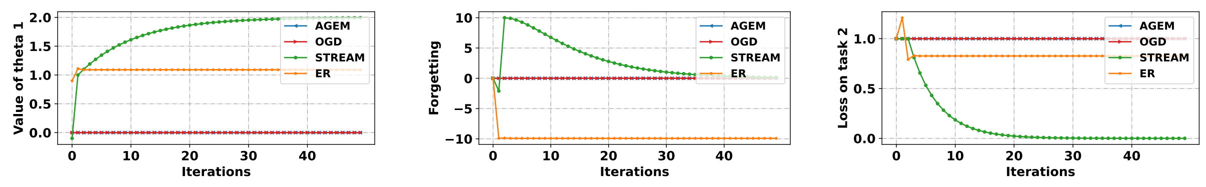
(b) AVE-CI, $p = 0.2$

🟦 EWC ⬜ MAS 🟧 AGEM 🟦 OGD 🟪 DER 🟫 GDumb 🟥 MEGA 🟪 AV-CIL 🟩 STREAM

Figure 5. Performance (test accuracy and forgetting) of continual learning methods on AVE-CI over 5 runs

not methods with high performance in class incremental scenarios, mainly due to the approximations mentioned by the authors. Currently, better-performing methods include those that use the buffer as a replay strategy, similar to STREAM. However, the authors do not mention any of the replay methods (ER, DER, among many others).

- A proper evaluation could be done with Experience Replay (ER) or other methods that replay previous examples during training of the new task together with the current task.



(a) The value of the $\theta^{(1)}$ vs. iterations. (b) The forgetting vs. iterations. (c) Loss on task 2 vs. iterations.

Figure 6. Synthetic experiment for the counterexample. (a), (b), (c) show the evolution of the value of $\theta^{(1)}$, the forgetting on task 1 and the loss on task 2. STREAM can find the optimal $\theta^{(1)}$ and achieve minimal forgetting and loss on the new task. But A-GEM and OGD fail to update their parameter $\theta^{(1)}$ toward the optimum throughout the training process, thus cannot minimize the loss on the new task. ER minimizes the loss functions f_1 and f_2 jointly (Assume that memory is large enough, it can visit f_1 as it needs). ER cannot find the the optimal value of $\theta^{(1)}$, thus can not achieve the low loss on task2. ER performs well in terms of forgetting, but STREAM still exhibits good forgetting metric (forgetting = 0).

A1. Thank you for your insightful question. Now we investigate the behavior of experience replay (ER) algorithm on our counterexample. ER algorithm trains the model on the current task and the buffer data jointly. In our counterexample, it can revisit the task 1 (objective function $f_1(\theta)$) when minimizing the current task 2 (objective function $f_2(\theta)$). Then the objective function can be reformulated as

$$\min_{\theta} f(\theta) := f_1(\theta) + f_2(\theta) = \begin{cases} 10(\theta^{(1)}-1)^2 + (\theta^{(1)}+1)^2 & \text{if} \quad \theta^{(1)} < 1/2, \\ 10(\theta^{(1)}-1)^2 + (\theta^{(1)}-2)^2 & \text{if} \quad \theta^{(1)} \geq 1/2. \end{cases}$$

Giving the initial point $(0,0)$, the optimal solution we can get from the ER objective function is $(12/11, a)$ for some $a \in \mathbb{R}$ (one can take the derivative w.r.t. $\theta^{(1)}$ and make it equal to 0 to find the optimal solution). However, this solution is suboptimal for the constrained optimization problem (whose optimal solution is $(2, a)$ for some $a \in \mathbb{R}$, and this solution achieves minimal possible error for the second task with zero forgetting).

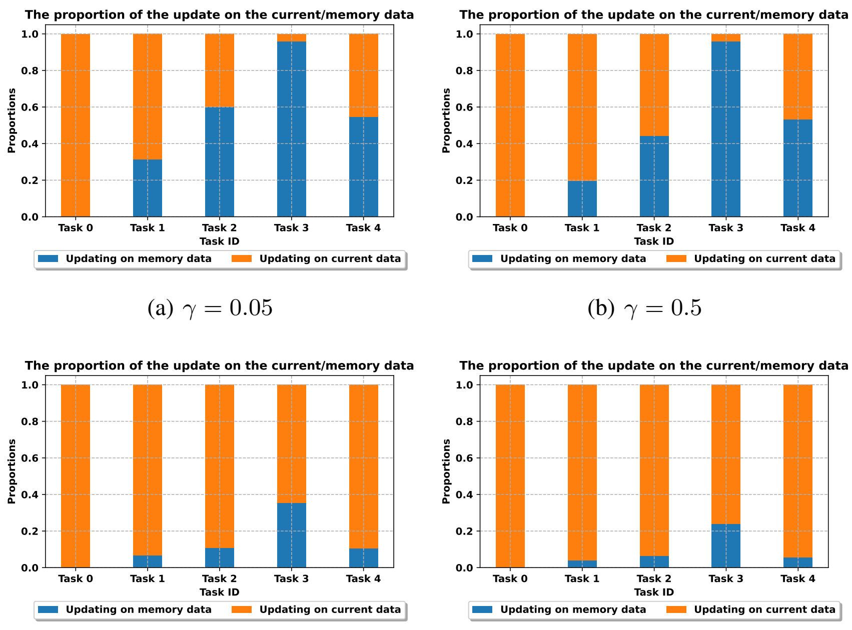
We empirically implement ER on counterexample in gradient descent with the step size $\eta = 0.05$, which keeps the same as section 6.1, and show the results in Figure 6. Consistent with the analysis, ER reaches the point $\theta^{(1)} = 12/11$, which is not the optimal point, so the loss on the new task, i.e., f_2, can not be minimized to reach the loss value 0. But it achieves the best forgetting performance among these algorithms. Notably, all the algorithms in this example can achieve good forgetting (≤ 0). But only STREAM achieves the best possible learning performance on the current task without any forgetting.

Q2. Although the method is well-motivated and it has a clear theoretical background. STREAM is just re-inventing ER in that it uses the samples from the buffer to update the weight directly. The only difference with ER is that they propose to iterate between training from the buffer and training from the current task, using the threshold to guarantee forgetting. However, this guarantee can significantly affect learning the new distribution, depending on the threshold.

A2. We would like to emphasize that our proposed method is different from ER among the following aspects. First, our problem formulation is different from theirs. For ER, they reformulate the CL problem to a unconstrained optimization problem, where the objective is written as the sum of the loss on the training data from the current task and the loss on the memory data. In contrast, we consider the original CL formulation, which is a constrained optimization problem (see eq. (1) in our paper), where the objective is the loss evaluated on the training data from the current task and the constraint is the loss evaluated on the memory data. Directly solving the original CL problem (with theoretical guarantee) is more challenging than the reformulation considered in ER, since nonconvex constrained optimization is harder than unconstrained optimization. Second, for each iteration, ER sample both the training data and memory data to make update, while we sample either the training or memory data to make update depending on the forgetting threshold.

In practice, we acknowledge that the forgetting threshold γ is indeed crucial to the performance of STREAM. Nevertheless, setting γ to be small within some range (e.g., $\gamma \leq 0.1$) works well for all settings we have tried in the experiments. This indicates that our method is not sensitive but robust to the forgetting threshold γ.

Q3. Did you study how many iterations the model (in an epoch) trained from the buffer versus the current task? I assume that if the threshold is big, the model will train more from the current task; however, if the threshold is small enough, the model will train more from the buffer. This can impact the amount of iteration that the model trains.



(a) $\gamma = 0.05$ (b) $\gamma = 0.5$

(c) $\gamma = 0.7$ (d) $\gamma = 1.0$

Figure 7. The frequency statistics of model update on the current/memory data.

A3. Actually your intuition is correct. We explored the impact of different forgetting thresholds on the model performance in Appendix A.4.2 and show the results in Figure 9 in the original submission ("epsilon" is the forgetting threshold in Figure 9, will be corrected to γ in the new version). Also, we collected statistics of the model update on the current/memory data for each task. The experiment is executed on Multiple Dataset, which consists of 5 tasks. There are 25 training epochs in each task, so we record the updating counts of iterations on the current data and memory data in each epoch within a task, respectively. Then the proportions of update on current/memory data for each task at different forgetting thresholds are illustrated in Figure 7. The model just learns the current data in the first task since the memory buffer is empty in the first task. As the model encounters more new tasks (from the task 0 to the task 3), the forgetting becomes more severe, which increases the updating proportion on the memory data. The model improves the performance on the memory data in the third tasks, such that it reduces the update frequency on the memory buffer and switches to the current data in task 4.

The forgetting threshold affects the model update dynamics, i.e., larger threshold improves the update frequency on the memory buffer. As you can see in Figure 7, the update frequency on the memory buffer reduces greatly when the forgetting threshold is large (i.e., $\gamma > 0.5$). Consequently, too large thresholds degrade model performance, as illustrated in Figure 9 in the original submission. Therefore, we prefer to choose a small forgetting threshold in an appropriate range empirically.

Q4. In Figure 3.b (and others), is there an intuition for the decrease in performance of STREAM in Task 2?

A4. Thanks for your insightful question. The intuition is that, at the beginning, the moving-average estimator (line 8 in Algorithm 1) does not approximate the forgetting measure very accurately, and therefore the right balance of two update branches may not be obtained perfectly. This causes the decrease in performance of STREAM in Task 2. As more moving-average steps (i.e., more iterations of the algorithms) are executed, the forgetting measure is approximated more accurately, and the model performance gradually increases.

Q5. In line 100, ".. but practical CL algorithms are single loop and only use small batch size to learn from continuous data streams". Why mention this? The scenario and methods give the size of the batch size and other characteristics. There are scenarios where higher batch size is permitted or when algorithms choose to do multiple loops, like meta-learning approaches.

A5. Thanks for your insightful question and we are sorry for the ambiguity. Please note that here we view batch size requirement (whether large or small) from a theoretical perspective, instead of meaning the actual number of batch size in experiments. To be more specific, for any given small $\epsilon > 0$, we refer "large batch size" to that depends on $1/\epsilon$, and we refer "small batch size" to that does not depend on $1/\epsilon$, that is, batch size can be $O(1)$.

We mention "small batch size" in order to emphasis that our proposed single loop algorithm works for any batch size in theory. For Theorem A.14 in (Huang & Lin, 2023), the batch size is set to be $B = \tilde{O}(1/\epsilon^4)$, which is obviously not practical when ϵ is sufficiently small, while we don't pose any requirement on batch size in our algorithm design and analysis. In particular, we introduce the momentum-style moving-average estimator (line 8 in Algorithm 1) to approximate the forgetting measure at every iteration, such that we could remove the need for large batch size theoretically.

Q6. In Algorithm 1, what is K?

A6. The number of iterations K in Algorithms 1 denotes the number of epochs multiplied by the number of data batches. Please see Appendix A.1 for more details.

Q7. In Algorithm 1, when discarding samples, which samples are discarded? Those added first (FIFO) or randomly selected? This can significantly impact the performance of the memory-based methods.

A7. Thank you for your insightful question. In Algorithm 1, we randomly select and discard samples.

Q8. Why use the selected hyper-parameters? For example, the memory size is really small. In the case of Tiny-Imagenet, where there are 200 classes, the memory size of 256 means less than 2 elements per class.

- Were the best hyper-parameters for each method found? Or did the ones found to optimize STREAM used in all methods?

  A. There are some common and important hyper-parameters in CL, including the memory size, the number of tasks, the noise level (in our setting). For fair comparison, we set memory size to $\{64, 256, 256, 340\}$ for all the algorithms on four benchmarks respectively. Actually, these hyperparameters are not tuned for the best performance of STREAM, instead they are commonly adopted in CL papers, we also compare these algorithms with different memory size, i.e., $\{m = 64, 128, 256\}$ on Multiple Dataset (noise-free) and $\{m = 256, 512, 1024\}$ on Split Tiny-Imagenet (noise-free). the results are presented in Table 4 and 5. We also explored the impact of different number of tasks, i.e., $\{T = 10, 20, 25\}$ on Split CIFAR-100 (noise-free) in Table 6. STREAM performs consistently the best among different memory sizes and number of tasks.

- How many seeds were used in the experiments?

  A. Random seed $\{0, 1, 2, 3, 4\}$ are used in our experiments.

- Do you have an intuition for the low performance of DER? Normally its performance is higher than AGEM.

  A. We did notice that DER performs not well in noise-free Split CIFAR-Imagenet in Table 4 (Appendix A.2 in the original paper). The reason mainly stems from the task construction and training epochs, there are 20 tasks in Split Tiny-Imagenet, instead of 10 as the DER paper mentioned. The number of training epochs (epochs=10) is much fewer than the orginal DER paper (epochs=100). Fewer training epochs are encouraged in the field of contintual learning. More sequentail tasks and fewer training epochs cause lower performance of DER. But DER (test acc 20.34 ± 0.51) performs better than A-GEM (test acc 18.61 ± 0.39) under the noise setting $p = 0.2$ since noise greatly interferes the gradient projection of A-GEM. When the difference of data distribution between different tasks is large, like standard Multiple Dataset, DER (test acc 59.25 ± 4.52) surpasses A-GEM (test acc 33.47 ± 1.61) in a large margin. On multi-model case, DER also (test acc 38.07 ± 0.72) outperforms A-GEM (test acc 31.22 ± 0.83) significantly.

Table 4. Results vs. memory size (m is the memory size) on Multiple Dataset.

Methods	$m = 64$		$m = 128$		$m = 256$	
	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
EWC	26.50±1.66	0.377±0.013	26.67±1.40	0.245±0.017	26.59±0.11	0.223±0.008
MAS	32.46±3.83	0.538±0.181	33.54±1.47	0.538±0.041	34.15±1.63	0.536±0.039
AGEM	33.47±1.61	0.541±0.179	36.04±0.94	0.480±0.005	41.90±1.52	0.424±0.026
OGD	27.88±1.23	0.375±0.015	32.24±1.34	0.544±0.051	33.63±1.24	0.303±0.022
DER	59.25±4.52	0.273±0.106	59.78±0.94	0.225±0.015	60.53±2.67	0.240±0.028
GDumb	53.03±0.34	**0.032±0.026**	58.64±0.42	**0.000±0.006**	69.11±2.83	**0.000±0.018**
MEGA	63.36±3.51	0.210±0.109	66.44±2.03	0.292±0.015	67.24±2.38	0.214±0.003
STREAM	**72.08±1.40**	0.152±0.035	**74.14±2.01**	0.145±0.016	**74.07±0.95**	0.025±0.048

Table 5. Results vs. memory size (m is the memory size) Split Tiny ImageNet.

Methods	$m = 256$		$m = 512$		$m = 1024$	
	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
EWC	16.79±0.74	0.311±0.010	17.15±0.44	0.306±0.002	17.05±0.10	0.324±0.003
MAS	15.51±0.89	0.312±0.010	17.03±0.08	0.332±0.008	18.56±0.52	0.316±0.000
AGEM	26.22±0.36	0.210±0.003	27.95±0.58	0.217±0.004	32.97±0.15	0.181±0.004
OGD	15.21±0.53	0.325±0.008	16.53±0.67	0.357±0.007	17.15±0.52	0.347±0.005
DER	18.00±0.76	0.262±0.010	21.46±1.13	0.236±0.005	23.10±0.51	0.214±0.007
GDumb	27.15±0.83	**0.035±0.001**	32.81±0.25	**0.040±0.004**	33.51±0.14	**0.021±0.002**
MEGA	29.30±0.38	0.148±0.002	32.87±0.63	0.137±0.007	33.20±0.44	0.104±0.001
STREAM	**31.36±0.71**	0.121±0.008	**33.02±0.53**	0.115±0.003	**34.67±0.14**	0.104±0.002

Table 6. Results vs. the number of tasks (T denotes the number of tasks) on Split CIFAR-100.

Methods	$T = 10$		$T = 20$		$T = 25$	
	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
EWC	27.98±1.57	0.403±0.013	48.56±2.16	0.473±0.014	47.43±3.23,	0.530±0.033
MAS	31.40±2.34	0.380±0.021	34.90±1.54	0.438±0.009	31.68±2.16	0.496±0.022
AGEM	39.72±2.43	0.292±0.019	54.02±1.61	0.251±0.013	48.19±1.65	0.345±0.019
OGD	29.13±1.52	0.262±0.025	34.19±1.57	0.451±0.019	36.35±1.43	0.353±0.023
DER	41.99±1.80	0.264±0.018	46.05±1.29	0.324±0.011	49.65±0.90	0.334±0.008
GDumb	38.80±1.08	**0.058±0.002**	55.85±0.46	**0.036±0.006**	67.56±0.53	**0.008±0.009**
MEGA	48.03±1.44	0.176±0.003	61.74±0.77	0.156±0.010	68.55±0.57	0.094±0.007
STREAM	**50.33±0.66**	0.167±0.008	**64.06±0.86**	0.132±0.010	**69.70±0.37**	0.080±0.010

5. Reviewer UdRx (rating 4 confidence 3)

Q1. The claim that ”existing memory-based CL methods do not work” for the counterexample is too strong. The authors only consider three classical memory-based CL methods, whereas there are a lot of memory-based methods proposed every year. The fact that these three methods do not work does not imply other existing memory-based methods do not work.

A1. Thank you for your suggestion and we will modify the statement in the revised version. We will change it to ”some representative memory-based CL methods, including A-GEM, GEM, OGD, ER, fail on our constructed counterexample.” Now we investigate the behavior of experience replay (ER) algorithm on our counterexample. ER algorithm trains the model on the current task and the buffer data jointly. In our counterexample, it can revisit the task 1 (objective function $f_1(\theta)$) when minimizing the current task 2 (objective function $f_2(\theta)$). Then the objective function can be reformulated as

$$\min_{\theta} f(\theta) := f_1(\theta) + f_2(\theta) = \begin{cases} 10(\theta^{(1)} - 1)^2 + (\theta^{(1)} + 1)^2 & \text{if} \quad \theta^{(1)} < 1/2, \\ 10(\theta^{(1)} - 1)^2 + (\theta^{(1)} - 2)^2 & \text{if} \quad \theta^{(1)} \geq 1/2. \end{cases}$$

Giving the initial point $(0, 0)$, the optimal solution we can get from the ER objective function is $(12/11, a)$ for some $a \in \mathbb{R}$ (one can take the derivative w.r.t. $\theta^{(1)}$ and make it equal to 0 to find the optimal solution). However, this solution is suboptimal for the constrained optimization problem (whose optimal solution is $(2, a)$ for some $a \in \mathbb{R}$, and this solution achieves minimal possible error for the second task with zero forgetting).

We empirically implement ER on counterexample in gradient descent with the step size $\eta = 0.05$, which keeps the same as section 6.1, and show the results in Figure 6. Consistent with the analysis, ER reaches the point $\theta^{(1)} = 12/11$, which is not the optimal point, so the loss on the new task, i.e., f_2, can not be minimized to reach the loss value 0. But it achieves the best forgetting performance among these algorithms. Notably, all the algorithms in this example can achieve good forgetting (≤ 0). But only STREAM achieves the best possible learning performance on the current task without any forgetting.

Q2. The baseline methods compared in the experiments are outdated. On standard single modality CL benchmarks, the oldest baseline was published in 2020. Considering that so many advanced memory-based approaches (including replay-based and orthogonal-projection based approaches) have been proposed after that, the current experimental results are not sufficient to justify the superiority of the proposed method.

A2. Thanks for your valuable suggestion, we find two popular related work: OCS (Yoon et al., 2022) and MetaSP (Sun et al., 2022), where OCS proposes an online coreset selection method for buffer data by maximizing several similarity metrics based on data pairs within each minibatch, and sample pairs between each minibatch and coreset. MetaSP introduces the concept of the example influence for CL model, and develops a meta-based methods to compute example influence, and update the buffer data based on the influence. For fair comparison, we fix the memory buffer to 64 for Multiple Dataset and 256 for Tiny-Imagenet. For OCS, the coreset size is an important hyperparameter, which is the size of the “core” samples in a mini-batch streaming data. Coreset size is smaller than mini-batch size, so we follow their original setting and set to 10, and balanced hyperparameters $\tau = 1000$ and $\lambda = 0.5$. For MetaSP, there is a addition validation set called Stability-Plasticity (SP) for example influence compurting. Following their original paper, we construct the SP validation set by randomly sampling 10% of the seen data and 10% of the memory buffer at each training iteration. Other hyperparameter settings follows the Table 1 in Appendix of the submitted paper. The results of noise-free ($p = 0.0$) and noise ($p = 0.2$) are summaried in Table 7. In terms of test accuracy, STREAM outperforms OCS and MetaSP in the noise-free setting and enlarges the accuracy gap in a large margin in the noise setting. The noise in fact has an negative influence to sample selection, leading to the suboptimal coreset or samples getting into the memory buffer.

Q3. In the counterexample, why is the model parameter set to be $(0, 0)$ after learning the first task? Would this value affect the effectiveness of the three memory-based methods on this example?

A3. Thanks for your insightful question. We would like to emphasize that the model parameters were set to be $\theta^{\text{init}} = (0, 0)$ after the first task merely for simplicity. This value does not affect the efficacy of the three memory-based methods in our counterexample. In fact, we can show that these three methods would not work regardless of the initial points from which we start at the beginning of task 2.

We consider the general case where the model parameter is set to be $\theta^{\text{init}} = (a, b)$ after the first task, the memory gradient is $\nabla f_1(\theta^{\text{init}}) = (20a - 20, 0)$ and the gradient on the current task is $\nabla f_2(\theta^{\text{init}}) = (2a + 2, 0)$ or $\nabla f_2(\theta^{\text{init}}) = (2a - 4, 0)$.

Table 7. Results of task-incremental experiments on Multiple Dataset and Split Tiny-Imagenet (p denotes the noise rate).

	Methods	$p = 0.0$		$p = 0.2$	
		ACC (↑)	FGT (↓)	ACC (↑)	FGT (↓)
Multiple Dataset	OCS	55.65±2.26	**0.062±0.001**	45.03±4.16	**0.049±0.012**
	MetaSP	57.14±1.10	0.113±0.042	47.14±1.66	0.081±0.027
	STREAM	**72.08±1.40**	0.152±0.035	**73.50±2.25**	0.130±0.037
Split Tiny-Imagenet	OCS	41.29±0.09	**0.112±0.001**	35.36±0.94	**0.061±0.005**
	MetaSP	43.33±0.32	0.127±0.002	37.18±0.76	0.068±0.007
	STREAM	**47.92±0.45**	0.153±0.018	**43.32±0.02**	0.367±0.013

Therefore, following the same procedure as in Section 4, we can obtain $\mathtt{Proj}^{\perp}_{\nabla f_1(\theta^{\text{init}})}\nabla(f_2(\theta^{\text{init}})) = (0, 2a+2)$ or $\mathtt{Proj}^{\perp}_{\nabla f_1(\theta^{\text{init}})}\nabla(f_2(\theta^{\text{init}})) = (0, 2a-4)$. Then these three methods are only able to update the second coordinate of the model parameter during the learning process, which means that these algorithms can only find a solution of type $(0, c)$ for some $c \in \mathbb{R}$, which is suboptimal and cannot learn any new information about the new task.

Q4. How did you calculate the gradient projection in the counterexample?

A4. Thanks for your insightful question. We would like to emphasis that the projection step in Algorithm 1 is introduced mainly for theoretical analysis. In theory, we need $\mathcal{K}$ to be convex and compact with the projection step to leverage the high probability lemmas (e.g., Lemma B.3, B.5 and B.8). In our counterexample, take task 2 for instance, we let $\mathcal{K}$ to be the ball centered at the initial point of task 2 (i.e., $\theta_0^2 = (0,0)$) with radius $R = 2.5$, and we compute the projection step via

$$\hat{\theta}_{k+1}^t = \theta_k^t - \eta\nabla_\theta \ell(\theta_k^t; \zeta_k^t) \qquad \text{and} \qquad \theta_{k+1}^t = \min\left\{1, \frac{R}{\|\hat{\theta}_{k+1}^t\|}\right\} \cdot \hat{\theta}_{k+1}^t,$$

and it works well in our experiments.

Q5. In terms of the difference between the proposed algorithm and existing methods for solving constrained optimization, the first challenge mentioned in section 5 is that the existing methods are developed for convex or smooth settings. However, I didn't see how the convexity or smoothness would play a role in your algorithm design. That said, why can't the existing methods be applied here if you use subgradient to replace the gradient therein?

A5. Thanks for your insightful question. (Polyak, 1963) focused on constrained convex optimization, while our objective function is neural network, which is generally nonconvex. (Zhang et al., 2022) studied deterministic smooth nonconvex optimization problem with convex constraints, while we consider a more general setting where the subgradients and function values of the objective and constraint can only be accessed through stochastic oracles. Thus those two methods above can not be applied to our setting even if we replace the gradient with subgradient.

(Ma et al., 2020) addresses stochastic non-convex constrained optimization problem using a double loop method in their Algorithm 3, which tackles the same setting as ours. However, their double loop design is rather complicated: for each outer loop, it requires solving a non-trivial sub-problem in its inner loops, making it less practical than single loop algorithms.

Although convexity does not directly influence the algorithm design, it plays a crucial role in theoretical analysis. Also, we need to use subgradient and apply nonsmooth analysis, as we mentioned in Section 5.1, the objective function of the counterexample constructed in Section 4 is nonconvex and nonsmooth (e.g., function f_2 in counterexample is nonconvex and nonsmooth, where the nonsmooth point is at $\theta = (1/2, 0)$).

Q6. Following the previous question, how large is the batch size in (Huang & Lin, 2023) such that it can't be used in CL? It seems that the authors just applied some sort of variance reduction using moving average to handle the larger variance in smaller batch size. What is the double-loop design?

A6. Thank you for your insightful question. For any given small $\epsilon > 0$, the batch size is set to be $B = \widetilde{O}(1/\epsilon^4)$ per Theorem A.14 in (Huang & Lin, 2023), while we don't pose any requirement on batch size in our algorithm design and analysis. In other words, our theory works for any batch size. We indeed introduce the momentum-style moving-average estimator (line 8 in Algorithm 1) to approximate the forgetting measure at every iteration, such that we could remove the need for large batch size. It's worth noting that this is different from variance reduction since we don't use any variance reduction-based update on the subgradient.

For double loop design, we refer the reviewer to Algorithm 3 in (Ma et al., 2020) for more details. It is more complicated and less practical than single loop methods since for each outer loop, it needs to solve a sub-problem in inner loops.

Q7. How did you choose β and γ? It seems that they should be chosen jointly since the estimation accuracy of the constraint value (depending on β) clearly affects the conservatism in the threshold selection (value of γ).

A7. Thank you for your insightful question. In theory, for sufficiently small $\epsilon > 0$, we choose $1 - \beta = O(\epsilon^4)$ and $\gamma = O(\epsilon^2)$, please see Theorem B.10 in Appendix B for more details, so basically β is large (close to 1) and γ is small (close to 0). In practice, we set $\beta = 0.99$ and set γ to be small within some range (e.g., $\gamma \leq 0.1$ works well).

Q8. What about the training cost comparison?

A8. To assess the running time of various algorithms, each one was executed independently on a device (NVIDIA A6000 48GB GPU). The elapsed wall-clock time was recorded throughout the entire training process, which encompassed 25 epochs on the standard (noise-free) Multiple Dataset and 10 epochs on Split CIFAR-100. The evaluation results are summarized in Table 8. Notably, STREAM emerges as the algorithm with the fastest training speed. Unlike other algorithms using the mixed gradient (current gradients and memory gradients) to update model parameters for each update iteration, STREAM adopts switching gradient which would chooses strategically one of gradients (either current gradients or memory gradients) to implement the backpropagation in each iteration. Consequently, this renders STREAM algorithm significantly more time-efficient.

Table 8. Running time on Multiple Dataset and Split CIFAR-100.

Methods	Multiple Dataset (hours)	Split CIFAR-100 (hours)
EWC	0.16	1.31
MAS	0.17	1.31
AGEM	0.16	1.30
OGD	0.47	3.01
DER	0.18	1.27
GDumb	0.13	0.92
MEGA	0.15	1.05
STREAM	0.11	0.79

Q9. There is a typo in eq. (6). ζ_k^t should be sampled from the memory instead of the current task data.

A9. Thank you for catching this issue. We will fix the typo in the revised version.

References

Caccia, L., Aljundi, R., Asadi, N., Tuytelaars, T., Pineau, J., and Belilovsky, E. New insights on reducing abrupt representation change in online continual learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=N8MaByOzUfb.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Cutkosky, A. Anytime online-to-batch, optimism and acceleration. In *International conference on machine learning*, pp. 1446–1454. PMLR, 2019.

Farajtabar, M., Azizan, N., Mott, A., and Li, A. Orthogonal gradient descent for continual learning. *arXiv preprint arXiv:1910.07104*, 2019.

Guo, Y., Liu, M., Yang, T., and Rosing, T. Improved schemes for episodic memory-based lifelong learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Huang, Y. and Lin, Q. Single-loop switching subgradient methods for non-smooth weakly convex optimization with non-smooth convex constraints. *arXiv preprint arXiv:2301.13314*, 2023.

Lin, S., Ju, P., Liang, Y., and Shroff, N. Theory on forgetting and generalization of continual learning. *arXiv preprint arXiv:2302.05836*, 2023.

Ma, R., Lin, Q., and Yang, T. Quadratically regularized subgradient methods for weakly convex optimization with weakly convex constraints. In *International Conference on Machine Learning*, pp. 6554–6564. PMLR, 2020.

Polyak, B. T. Gradient methods for minimizing functionals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 3 (4):643–653, 1963.

Sun, Q., Lyu, F., Shang, F., Feng, W., and Wan, L. Exploring example influence in continual learning. *Advances in Neural Information Processing Systems*, 35:27075–27086, 2022.

Yoon, J., Madaan, D., Yang, E., and Hwang, S. J. Online coreset selection for rehearsal-based continual learning. In *International Conference on Learning Representations*, 2022.

Zhang, J., Pu, W., and Luo, Z.-Q. On the iteration complexity of smoothed proximal alm for nonconvex optimization problem with convex constraints. *arXiv preprint arXiv:2207.06304*, 2022.