Poisoning Generative Replay in Continual Learning to Promote Forgetting

Siteng Kang [1] **Zhan Shi** [1] **Xinhua Zhang** [1]

Abstract

Generative models have grown into the workhorse of many state-of-the-art machine learning methods. However, their vulnerability under poisoning attacks has been largely understudied. In this work, we investigate this issue in the context of continual learning, where generative replayers are utilized to tackle catastrophic forgetting. By developing a novel customization of dirty-label input-aware backdoors to the online setting, our attacker manages to stealthily promote forgetting while retaining high accuracy at the current task and sustaining strong defenders. Our approach taps into an intriguing property of generative models, namely that they cannot well capture input-dependent triggers. Experiments on four standard datasets corroborate the poisoner's effectiveness.

1. Introduction

The vulnerability of machine learning systems must be scrutinized before they can be deployed to security-critical applications. The common evasion attack assumes that clean target instances can be manipulated at test time, which can be unrealistic in many scenarios. In contrast, poisoning attacks only make malicious and imperceptible modifications to the training set, so that the prediction on test examples can be mistaken. The threat models may insert poison examples (Chen et al., 2017), flip the training labels (Xiao et al., 2012; Levine & Feizi, 2021), or modify the training example inputs (Biggio et al., 2012; Shafahi et al., 2018).

Although poisoning attacks have been extensively studied under discriminative learning, their potential risk in *generative* learning has been largely understudied. Ding et al. (2019) poisons the training examples so that the learned generator covertly changes some important part of the output image, e.g., turning a red light into green. Salem et al. (2020) enables the adversary to control the output image by planting a trigger in the input image or noise. Both are backdoor attacks requiring write access to test data, and work in batched learning scenarios.

The increasing penetration of generative models in machine learning urges the investigation of poisoning attacks in a broader range of learning paradigms. In this work, we focus on continual learning, a prominent setting where tasks arrive in streams and each of them corresponds to a discriminative learning problem such as classification (Chen & Liu, 2018). Since the tasks are streamed and cannot be stored, the running classifier often suffers from *catastrophic forgetting*, where the performance on older tasks gradually deteriorates (McCloskey & Cohen, 1989).

Deep generative replay (DGR) is a natural tool to bring back the memory of the previous tasks by learning a generative model to fit the data of these tasks (Shin et al., 2017; Cong et al., 2020). Despite their effectiveness, new vulnerabilities are also opened up where misleading examples can be injected to the training data $\mathcal{D}_t$ for the current task t, so that catastrophic forgetting can be *promoted* when such poisoned $\mathcal{D}_t$ is used for training both the replayer G_t and the classifier. In this work, we seek practical and stealthy poisoning attacks on DGR that achieve three objectives:

- O_{eff} (effective): After moving past task t, the classifier will soon forget what was learned from it (i.e., perform poorly on clean test examples drawn from it) despite using a replayer for all the tasks seen so far.
- O_{ste} (stealthy): *During* task t, the classifier trained from poisoned data does *not* suffer degradation of test accuracy on task t itself. This is important because poor performance on the current task raises immediate suspicion. In contrast, by promoting forgetting, the harm will manifest itself only after the victim has moved on to the next task, by which time it will have become too late because access to the samples of task t is already lost.
- O_{rob} (robust): The poisoned data should be robust to solid defenses deployed by both the classifier and the replayer.

The main difficulty lies in two folds. Firstly, although both O_{eff} and O_{ste} are straightforward to fulfill individually, they are at odds with each other and are hard to fulfill simultaneously. Secondly, since the tasks are streamed, the adversary

[1]Department of Computer Science, University of Illinois Chicago. Correspondence to: Siteng Kang and Xinhua Zhang <{skang98,zhangx}@uic.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

must make irrevocable attacks at task t before future tasks arrive, i.e., before catastrophic forgetting takes place. Due to this difficulty, poisoning attacks have been much less studied in an online setting. Mladenovic et al. (2022) addressed the online decision problem of selecting k examples for evasion attack. Zhang et al. (2020) assumed the instances are drawn i.i.d. from a time-invariant distribution, which conflicts with continual learning. Other works require multiple passes of the data stream (Gong et al., 2019; Lin et al., 2017; Sun et al., 2020), or clairvoyant knowledge of future data (Burkard & Lagesse, 2017; Wang & Chaudhuri, 2018).

Our contribution, hence, is to overcome these challenges and to reveal the vulnerability of generative replayers – their training data can be poisoned stealthily such that a task can be learned well at present but be forgotten soon in the future. Noting that simple label-flipping poisoning can be easily detected, we resort to dirty-label backdoor/Trojan attack (Liu et al., 2018) to attain O_{ste}: the trained classifier performs correctly on clean examples, but errs if the example is planted with a trigger. To further achieve O_{eff} and O_{rob}, we capitalize on the input-aware backdoor (Nguyen & Tran, 2020), which allows the trigger to vary depending on the image. As a result, it can not only withstand stronger defense (§4), but also enjoys higher variation and stealthiness, hence much harder for a generative model to capture. So the replayed images do not well preserve the trigger (we call it trigger-discarding property in §3.3) while retaining the incorrect label, leading naturally to forgetting (§3). The problem is set up in §2, and experiments are provided in §5 to show the effectiveness of the attack. Our innovations are summarized as follows:

- Proposing the first poisoning attack that promotes catastrophic forgetting in continual learning.
- Achieving poisoning (no trigger is needed at test time) through a novel way of leveraging backdoor attack that is effective for exacerbating catastrophic forgetting.
- Identifying a trigger-discarding property of generative models that is intriguing for backdoor attack.

Related work Generative models have been pervasive in machine learning (Murphy, 2023, Part IV), reaching far beyond the original role of density estimation and serving as a key infrastructure in supervised, unsupervised, and reinforcement learning. We contend that their vulnerability needs to be examined in the context of their use. In the vanilla density estimation, Condessa & Kolter (2020) learned robust variational auto-encoder (VAEs) that retain high likelihood for the data points under adversarial perturbation. The underlying threat is evasion attack, and along similar lines, Tabacof et al. (2016) and Kos et al. (2018) studied attacks that promote reconstruction error of the decoder in a VAE. Some recent works address attacks on membership inference (Hayes et al., 2019; Chen et al., 2020; Hilprecht et al., 2019), model extraction (Hu & Pang, 2021), and attribute inference (Stadler et al., 2022). However, poisoning attacks on generative models are still understudied.

Our aim is to poison a generative model instead of learning a generative model to produce poisons for another (discriminative) model (Yang et al., 2017; Muñoz-González et al., 2019). We also leave it as future work to defend the proposed attack, noting that (certifiable) defense and detection have been well studied for poisoning attack on *batch* discriminative models (Peri et al., 2019; Steinhardt et al., 2017; Levine & Feizi, 2021; Jagielski et al., 2018).

2. Generative Models in Continual Learning

We consider the continual learning setting, where tasks arrive sequentially. The goal is to keep updating a classifier that predicts accurately not only on the current task, but also on the previous tasks. Each task t is indeed a joint distribution $P_t(X, Y)$, where $X \in \mathcal{X}$ is the input from a feature space $\mathcal{X}$, and $Y \in \mathcal{Y}_t$ is the label whose domain $\mathcal{Y}_t$ may change with the task. For example, $\mathcal{Y}_1$ consists of digits 0 and 1, and $\mathcal{Y}_2$ encompasses 2 and 3. Even in the case where the domains remain constant, the distribution P_t can shift. The goal of continual learning is to find a classifier C_t, such that the overall risk across all tasks seen *so far* is minimized:

$$C_t \approx \arg\min_C \sum_{i=1}^{t} \mathop{\mathbb{E}}_{(X,Y)\sim P_i} [\ell(C(X), Y)]. \tag{1}$$

Here ℓ is a loss function, and we will focus on multi-class classification with cross-entropy loss.

Unfortunately, the growing volume of data easily dwarfs the storage capacity, and other concerns such as privacy may even preclude storing past data altogether. So the performance on previous tasks may deteriorate significantly, a phenomenon known as *catastrophic forgetting*. For generality, we will follow many continual learning literature by considering the setting where **no data from past tasks is stored**. Among many solutions in such a regime, DGR-based approaches resort to learning a DGR model G_t that approximately replicates P_t. Then at each task t, samples of (X, Y) pairs are drawn from not only the current P_t, but also from the replayers for the previous tasks $G_{1:t-1} := \{G_i\}_{i=1}^{t-1}$. Their union is subsequently used to update the classifier into C_t. The whole process of vanilla DGR-based continual learning is illustrated in Algorithm 1.

DGR models can be simplified into a **single** running replayer G that is updated over time, as opposed to multiple replayers (one per task). However, our contribution is the **poisoner**, not the replayer. Employing multiple replayers only brings more challenges to our attacker, because we are now tasked to poison many of them, each of which can

Algorithm 1 DGR to combat forgetting (continual learning)

Input: Tasks $1, 2, \ldots$ represented as $P_1, P_2, \ldots$
1: Initialize classifier C_0
2: **for** $t = 1, 2, \ldots$ **do**
3: **for** $i \in [t-1] := \{1, 2, \ldots, t-1\}$ **do**
4: $\mathcal{S}_i \leftarrow$ **SampleFromDGR**(G_i) $(X, Y$ pairs$)$
5: Sample $\mathcal{D}_t$ from task P_t ▷ To be poisoned
6: $C_t \leftarrow$ **TrainClassifier**$(C_{t-1}, \mathcal{D}_t \cup \mathcal{S}_{1:t-1})$
7: $G_t \leftarrow$ **TrainReplayer**$(\mathcal{D}_t)$, e.g. conditional GAN

only be poisoned once at its current task. In contrast, if we simply keep a single running replayer, then we enjoy opportunities of *repeatedly* poisoning it at any time and our objectives will become much easier to achieve. To conclude, multiple replayers constitute a more stringent benchmark for testing our poisoner. A natural choice of the replayer is a conditional generative model such as conditional GAN (cGAN), which first samples the label Y from a discrete distribution, and then generates the feature X via a cGAN.

2.1. Attackers and learners

We first set up the two parties involved in the process. The **victim learner/user** consists of a *classifier*, a *replayer*, and a *defender*. We assume **none of them has access to the original clean data**, and can only access the poisoned data. Further, the replayer must perform well, i.e., the generated samples match the distribution of data presented to it for training. Otherwise, the replayer would not be adopted by the user in the first place, and can spare any need of attack by, e.g., generating random images with random labels.

A defender is employed by a learner to scrutinize and prune the possible poisons in the training data. In DGR, it means examining the data collected from the current task t, as well as the replayed samples from previous G_i in step 4. This is known as *pre-training* defense, whereas *post-training* defense patches up the learned model (Wang et al., 2019).

The **attacker** is *only allowed to poison (modify) the samples $\mathcal{D}_t$ in step 5 of Algorithm 1*. It has no access to the internal mechanism or weights of the classifier, replayer, or defender. Nor can it access the replayed data. Following the common practice such as Witches' brew (Geiping et al., 2021), we assume that the attacker internally trains a surrogate classifier $\hat{C}_t$ and a surrogate replayer $\hat{G}_t$ that closely reflect the architecture of the victim model (Goldblum et al., 2023), and queries them to construct the poisons. We will pursue a dirty-label backdoor attack, i.e., a small portion ρ_b of $\mathcal{D}_t$ will flip their label, along with a trigger of size ρ_a planted to their input image (more details in §3.2).

2.2. Achieving poisoning through backdoor

In continual learning, an attacker generally does not have the liberty of planting a trigger on test examples. So we will address in O_{eff} and O_{ste} a much more challenging setting (from an attacker's perspective) where such access is not available. It is important to note that although our method leverages backdoor attacks, it only serves as a means of achieving the overall goal, which is *poisoning* the generator/replayer instead of backdooring. In backdoor attacks, a classifier predicts poorly only on backdoored examples with triggers, while remaining well performing on clean test examples. In contrast, (non-backdoor) poisoning attacks aim to predict poorly on *all* examples, even without a "trigger".

2.3. Difficulties in the attack

Since most of the existing poisoning algorithms are in the batch setting, the extension to continual learning brings about new and significant challenges.

Firstly, the poison cannot compromise the *current* classifier, but should sufficiently poison the DGR so that the samples drawn from it during the *later* tasks will be detrimental enough to forget the previously learned tasks. **This rules out adding random images with random labels, or simply flipping the label of some examples in $\mathcal{D}_t$**, because they are easy to detect (Levine & Feizi, 2021) and the resulting classifier C_t will perform poorly on task t. Empirical demonstrations are available in Appendix G, where a baseline naive attack of flipping 25% labels on split-MNIST is considered, and the test accuracy of current tasks drops below 20% from the second task, failing the objective O_{ste}.

Secondly, since the future tasks have not been witnessed yet at task t and the training of C_{t+1} has not started, it is infeasible to optimize the forget-inducing distortion on $\mathcal{D}_t$ via back-propagation based optimization – the context and objective are not yet available for future forgetting.

3. The Attack Algorithm

Our poisoning attack proposes evading the defense by leveraging the *input-aware backdoor* attack (Nguyen & Tran, 2020), so that *mislabeled* data points carrying a *trigger* can be injected to $\mathcal{D}_t$ in a small amount. In particular, our approach achieves O_{eff}, O_{ste}, and O_{rob} through the following effects:

1. Since the triggers depend on (hence vary across) the input data, the defender can hardly detect it.
2. Since the mislabeled examples for the current task all carry an input-aware trigger, the learned backdoored classifier for task t makes mistakes only for backdoored examples. As such, it predicts accurately on pristine test examples for task t which carry no trigger. This will be

demonstrated in Section 5.1. For example, in Figure 1c, the first task (blue line) has nearly 100% accuracy with clean testing inputs on the first 100 epochs.

3. As generative models essentially represent a lossy compression, it is generally unable to capture the triggers that change with the input. When replayed later for a future classifier at task $t+1, t+2, \ldots$, the triggers go absent while the incorrect label is retained. So the classifier will be trained on mislabeled examples of task t with no backdoor, hence misclassifying clean test examples.

3.1. Backdoor attacks

Our solution is based on backdoor attacks, which despite the marked resemblance to poisoning attacks, do *not* misclassify a test example unless a pre-designed trigger is inserted to it. Backdoor attacks such as BadNets (Gu et al., 2019) plant a pre-selected or learned trigger into some training images at a pre-selected or varying location. A number of variations are available such as soft blending and multi-channel. Adding the resulting image to the training data along with a flipped label (randomly selected for an untargeted attack, and pre-specified for a targeted attack), a classifier can be trained that enjoys two important properties:

$\mathbf{P}_{\text{eff}}$ Once a test image is also backdoored with a trigger, the predicted label will change to the pre-specified (or random) one in a targeted (or untargetted) attack.

$\mathbf{P}_{\text{ste}}$ However, the test accuracy on pristine images (without a trigger) can remain very high.

Our inspiration originates from this trigger-based modulation. Suppose we plant the trigger on $\mathcal{D}_t$ in step 5 of Algorithm 1, and denote the resulting training set as $\tilde{\mathcal{D}}_t$. Then the resulting C_t will perform well on pristine test examples of task t, because they do not carry the trigger. After moving to task $t+1$, the replayer will (approximately) reproduce $\tilde{\mathcal{D}}_t$, at which point two cases can be considered:

- If the replayer works purely by rote, then $\tilde{\mathcal{D}}_t$ will be exactly replayed and the resulting classifier C_{t+1} will be backdoored in the same way as C_t. As a result, it will still predict accurately on clean samples from task t, i.e., the attack fails in promoting forgetting.
- If the replayer is lossy and is unable to capture or reproduce the trigger, then the replayed examples *might* no longer carry the trigger. However, they still carry the flipped label. As a result, C_{t+1} will now be trained on mislabeled examples without a trigger, and will therefore perform poorly on task t. In this case, the attacker successfully promoted forgetting.

The requirement on the replayer in the second case may appear unrealistic, because firstly the replayer is supposed to faithfully preserve the salient information in the inputs to address catastrophic forgetting. Secondly, a user (who constructs and trains the replayer) obviously has no motivation to collaborate with an attacker. Therefore, the key challenge for the attacker is to design delicate triggers that are *as likely to be overlooked and disregarded as possible* under generative modeling, while retaining the good performance on clean test data during task t (property $\mathbf{P}_{\text{ste}}$).

This is indeed challenging as we experimented. Static backdoor (BadNet) can be easily replayed by a generative model. Along with Trojan attack, it can be easily defended by neural cleansing. We also tested static backdoor with changing location, which again, turned out easily detected and fixed by neural cleansing. Witches' Brew (Geiping et al., 2021) and other gradient matching based methods need to know the target before deploying the attack, while future tasks are unknown in continual learning. Eventually, it turns out the *input-aware* backdoor satisfies our need, where a trigger is customized for each example through a learnable generative model, hence exhibiting much less regularity for the replayer to capture.

Algorithm 2 InputAwareBackdoor

Input: Data generation distribution $P(X, Y)$, which will the invoked with $P = P_t$ at task t

1: $(\tilde{\mathcal{D}}, \mathcal{L}_{div}) \leftarrow$ **InputAwareBackdoor–Obj**$(P, B_{[\mathcal{Y}]})$

Output: $\arg\min_{C, B_{[\mathcal{Y}]}}\{\mathcal{L}_{cl}(C) + \lambda_{div}\mathcal{L}_{div}\}$, where $\mathcal{L}_{cl}(C)$ is the classification risk $\sum_{(x,y)\in\tilde{\mathcal{D}}} \ell(C(x), y)$.

Algorithm 3 InputAwareBackdoor–Obj

Input: P as in **InputAwareBackdoor**, and backdoor generators $B_{[\mathcal{Y}]} := \{B_y : y \in \mathcal{Y}\}$

1: Initialize $\tilde{\mathcal{D}} = \emptyset$ which will contain clean and poisoned examples. Set $\mathcal{L}_{div} = 0$ (diversity loss).
2: **for** (x, y) sampled from P for task t **do**
3: sample $d \sim U(0, 1)$, sample y' from $\mathcal{Y}\backslash\{y\}$,
4: sample $(\hat{x}, \hat{y})$ from P excluding (x, y),
5: $\mathcal{L}_{div}$ += $\|x - \hat{x}\| \,/\, \|B_{y'}(x) - B_{y'}(\hat{x})\|$ ▷ diversity
6: **if** $d < \rho_b$ **then** ▷ make a backdoor example
7: $x' \leftarrow x \odot B_{y'}(x)$, $\tilde{\mathcal{D}}$ += (x', y')
8: **else if** $d < \rho_b + \rho_c$ **then** ▷ make a cross example
9: $x' \leftarrow x \odot B_{y'}(\hat{x})$, $\tilde{\mathcal{D}}$ += (x', y)
10: **else** $\tilde{\mathcal{D}}$ += (x, y) ▷ clean example

Output: $\tilde{\mathcal{D}}$ and $\mathcal{L}_{div}$

3.2. Input-aware backdoor

We first recap the input-aware backdoor (IAB, Nguyen & Tran, 2020) as shown in Algorithm 2 under a given data distribution P, and then detail how to utilize it for our purpose. Here, the classifier C and class-wise backdoor generating *networks* B_y are jointly optimized over an objective constructed in Algorithm 3, where each (x, y) sampled from P

contributes in one of the following modes:

- As a backdoor example with probability ρ_b: a wrong label y' is randomly picked, and then a trigger that depends on x is generated by $B_{y'}(x)$ and injected to x in line 7 via elementwise product $\odot$.
- As a cross-trigger example with probability ρ_c. To ensure that a trigger synthesized for one example x is *not* effective for another, a different $\hat{x}$ is sampled from P. Then x is poisoned with the trigger generated from $\hat{x}$ for a wrong label y', followed by pairing with the true label y (line 9).
- As a clean example otherwise (line 10).

In Algorithm 2, $B_{[Y]}$ is explicitized in line 1 to stress that both the diversity loss $\mathcal{L}_{div}$ and classification risk $\mathcal{L}_{cl}$ (through $\tilde{\mathcal{D}}$) are functions of $B_{[Y]}$, which is then optimized along with the classifier C. As observed in our experiment, IAB proffers the following property (Nguyen & Tran, 2020):

$\mathbf{P}_{\text{rob}}$ The backdoor in IAB can be hardly detected by state-of-the-art methods such as neural cleansing.

To summarize, we fulfilled O_{ste} by $\mathbf{P}_{\text{ste}}$, and O_{rob} by $\mathbf{P}_{\text{rob}}$. To meet O_{eff}, we require a *trigger-discarding* property as follows, which plays a key role in our method and will be discussed in the next subsection:

$\mathbf{P}_{\text{dis}}$ The replayer cannot well capture the trigger generation network of IAB, in the sense that the replayed examples do *not* well preserve the triggers.

3.3. Trigger-discarding generative models

Property $\mathbf{P}_{\text{dis}}$ depends on both the replayer and the trigger. If the trigger is a constant small white square at the image center, most standard generative models tested in our experiments managed to preserve it. Section 5.2 will show that static backdoors are preserved while the dynamic ones are dropped. Same is true if the replayer only replicates the training set. In general, it is supposed to capture the salient features of the input, and one might presume that triggers are likely to be discarded if they vary a lot across examples. It turns out not true. For example, we placed a square/triangle/round in random colors at random positions of the images, and a WGAN easily reproduced them with these (interpolated) color, shape, and position.

Formally, let P_x be the data distribution, and suppose given an input x, the trigger is generated by a learnable network $f_\theta(x)$ and is added to x by a pre-specified operation $g(x, f_\theta(x))$. It induces a distribution of backdoored examples as a push-forward of P_x: $Q_x^\theta := (x \mapsto g(x, f_\theta(x))) \# P_x$. Let a generative learning algorithm $\mathcal{A}$ map a set of examples $\{x_i\}_i$ to a distribution. Then the trigger is intended to *demote* some divergence (e.g., KL and Wasserstein) between

$$P_x \quad \text{and} \quad \mathbb{E}_{x_i \sim Q_x^\theta} \mathcal{A}(\{x_i\}_i). \tag{2}$$

Algorithm 4 Operation of the user, attacker, and defender during task t (in place of line 3 to 7 of Algorithm 1)

Input: P_t for task t, classifier C_{t-1}, replayers $G_{[t-1]}$, *surrogate* classifier $\hat{C}_{t-1}$, *surrogate* replayers $\hat{G}_{[t-1]}$, and backdoor generators $B_{[Y]}$

1: Initialize *surrogate* classifier $\hat{C}_t$ with $\hat{C}_{t-1}$.
2: **for** $i \in [t-1]$ **do** ▷ Sample from the *surrogate* replayer
3: $\quad \hat{\mathcal{S}}_i \leftarrow$ **SampleFromReplayer**$(\hat{G}_i)$ ▷ X, Y pairs
4: **for** number of iteration (**run in mini-batches**) **do**
5: $\quad \tilde{\mathcal{D}}_t, \mathcal{L}_{div} \leftarrow$ **InputAwareBackdoor–Obj**$(P_t, B_{[Y]})$ ▷ Both $\tilde{\mathcal{D}}_t$ and $\mathcal{L}_{div}$ are functions of $B_{[Y]}$
6: $\quad \mathcal{L}_{cl} \leftarrow$ sum of $\ell(\hat{C}_t(x), y)$ over $(x, y) \in \tilde{\mathcal{D}}_t \cup \hat{\mathcal{S}}_{1:t-1}$ ▷ $\mathcal{L}_{cl}$ is a function of $\hat{C}_t$ and $B_{[Y]}$
7: $\quad$ Update $\hat{C}_t$ to reduce $\mathcal{L}_{cl}$
8: $\quad$ Update $B_{[Y]}$ to reduce $\mathcal{L}_{cl} + \lambda_{div}\mathcal{L}_{div}$
9: Set $\tilde{\mathcal{D}}_t$ by using the learned $B_{[Y]}$.
10: Train a *surrogate* replayer $\hat{G}_t$ based on $\tilde{\mathcal{D}}_t$.
11: **for** $i \in [t-1]$ **do**
12: $\quad$ User: $\mathcal{S}_i \leftarrow$ **SampleFromReplayer**(G_i)
13: **Defender**: Apply ν-SVM on the replayed data $\mathcal{S}_{1:t-1}$
14: User: $C_t \leftarrow$ **TrainClassifier**$(C_{t-1}, \tilde{\mathcal{D}}_t \cup \mathcal{S}_{1:t-1})$
15: User: $G_t \leftarrow$ **TrainReplayer**$(\tilde{\mathcal{D}}_t)$
16: **Defender**: Neural Cleansing on C_t

Output: $C_t, G_t, \hat{C}_t, \hat{G}_t$, and $B_{[Y]}$

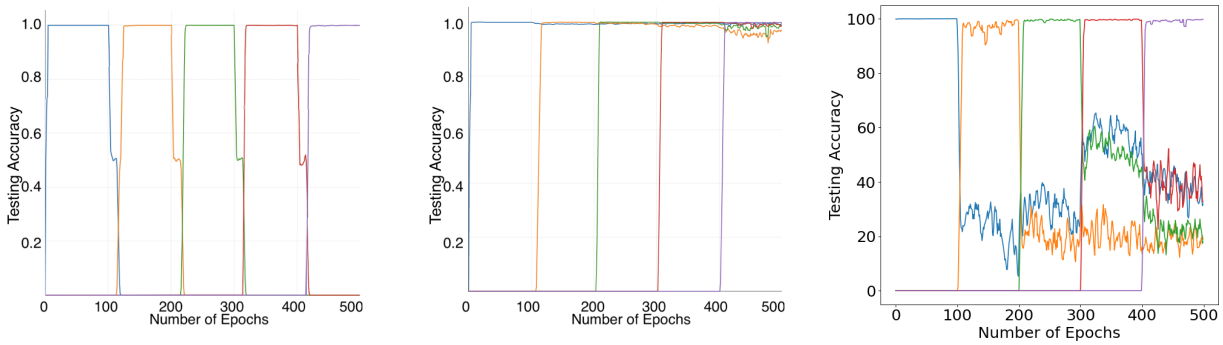
Directly computing the gradient in θ is expensive and infeasible, because $\mathcal{A}$ is assumed inaccessible in §2.1.

Fortunately, our experiments show that $\mathbf{P}_{\text{dis}}$ is well achieved (but not perfectly) when IAB is applied in conjunction with several SOTA generative models such as conditional Wasserstein GAN (cWGAN, Engelmann & Lessmann, 2021) and conditional VAE (Sohn et al., 2015). This is because these models have limited capacity, and the trigger's dependency on the input, which is more involved than just random, significantly increases sample complexity for generative learning. A theoretical analysis is left for future work, and §5.2 empirically illustrates this intriguing property.

3.4. Poisoning the replayer via IAB

We are now ready to apply IAB to poison the DGR and to promote forgetting. We will call our method Continual Input-Aware Poisoning (CIAP). It does *not* backdoor test images. Algorithm 4 demonstrates the operation of all participants during task t – attacker in black, user in blue, and defender in red (reserved for Section 4). It corresponds to the loop under a given t in Algorithm 1 (line 3 to 7 therein).

In line 6, the poisoned data $\tilde{\mathcal{D}}_t$ is joined with $\hat{S}_{1:t-1}$ from the surrogate replayers to construct the classification risk $\mathcal{L}_{cl}$. The attacker is then trained in the same way as in Algorithm 2, using a surrogate classifier $\hat{C}_t$. The surrogate



(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 1: Test accuracy on clean *test* images for **split-MNIST**

replayer $\hat{G}_t$ is additionally learned in line 10. The user trains its victim classifier and replayer in lines 14 and 15, respectively.

4. The Defender

We consider two defenses against the CIAP attack. The first is neural cleansing (Wang et al., 2019), which has been inserted in line 16 of Algorithm 4. If it were successful, then the backdoor planted in C_t would be detected and removed, thereby defeating our robustness objective O_{rob}.

Neural cleansing reverse engineers a universal trigger pattern that can turn every image into one specific label. Therefore, it will not work when the IAB attack enforced the non-reusability of triggers by having "cross" poisons. Nguyen & Tran (2020) provided more details and experiments, and we will thus not investigate neural cleansing in experiments.

Our second defense is aimed at objective O_{eff}. To this end, we apply an outlier detector ν-SVM to $\mathcal{S}_{1:t-1}$, which is in line 13 of Algorithm 4. If it managed to filter out mislabeled replayed samples, then the attacker would fail to bolster catastrophic forgetting. Here ν is a hyperparameter controlling the fraction of outliers. Since its value is unknown in practice, our experiment will enumerate a range of ν values, and demonstrate the extent to which the learner's performance can be saved respectively.

It is crucial to recognize that the replayed examples are *not* simply label-flipped poisons (i.e., clean images with a wrong label), although the replayer is poisoned with label-flipped and backdoored examples. This is for two reasons. Firstly, since the examples of a class y' is fed to the replayer to train for the class y, the generation of the features/images for class y is contaminated. Secondly, the input-dependent triggers introduce additional complications to the generative model. Indeed, we tested by directly generating label-flipped examples based on clean images, and ν-SVM easily filtered them out. However, this is not the case when ν-SVM is applied to our replayed images (§5.3).

5. Experimental Results

We next experiment on CIAP to verify: i) it attains the two objectives O_{eff} and O_{ste}; ii) the trigger-discarding property introduced in §3.3 holds true for commonly used generative models; iii) CIAP remains effective under strong defenders (O_{rob}). The code is available at Online Supplementary.

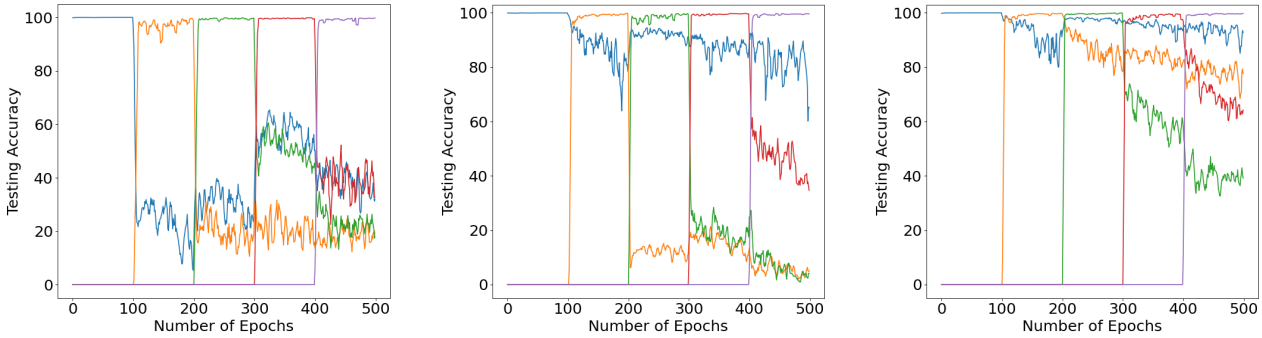
5.1. Effectiveness of the attack for O_{eff} and O_{ste}

We tested CIAP on five datasets: split-MNIST (Ciresan et al., 2011), split-CIFAR-10 (Krizhevsky & Hinton, 2009), FashionMNIST-MNIST (Xiao et al., 2017), permuted-MNIST (Goodfellow et al., 2014), and split-EMNIST (Cohen et al., 2017). We used SpinalVGG as the victim classifier (Kabir et al., 2020) for the four MNIST datasets, and ResNet (He et al., 2016) for split-CIFAR-10. The results shown here use cWGAN with gradient penalty as the replayer, and results for cVAE are deferred to Appendix F. The poison ratio $\rho_b = 0.25$, and the cross ratio $\rho_c = 0.15$. Each trigger was allowed to change 5% of the image's pixels.

split-MNIST We separated the entire dataset of MNIST into five tasks, each consisting of images from two disjoint classes in MNIST – the first task includes classes 0 and 1; the second task includes 2 and 3; and so on. The victim model was trained for 100 epoch in each task.

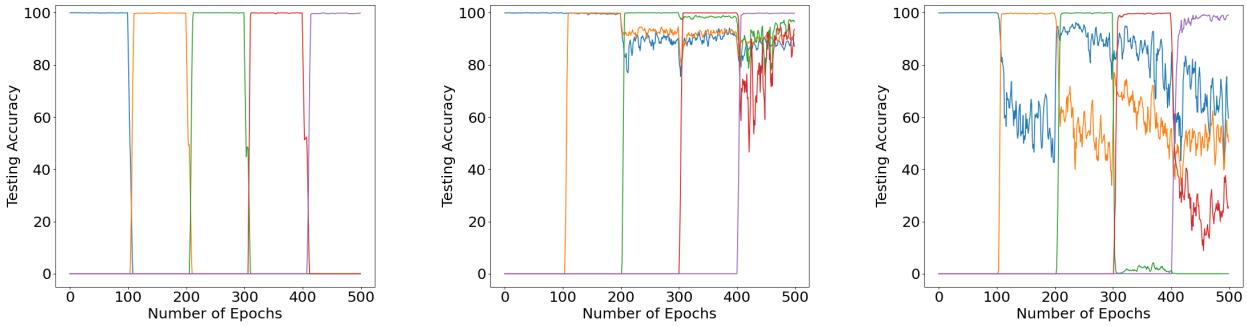
Figure 1a shows the baseline result without a replayer, where the blue line represents the test accuracy of the first task, orange line for the second, etc. As expected, the test accuracy for each task drops rapidly to 0% after the victim model proceeds to a new task. It is 0% because the new task has no overlap with the previous ones in the label space. Figure 1b shows the result of DGR-facilitated training, where the forgetting is significantly mitigated, and the test accuracy remains high on all trained tasks. This confirms the effectiveness of DGR and the sufficient capacity of the cWGAN.

Figure 1c shows the result after our attack CIAP is enacted. The test accuracy of each current task can still achieve nearly 100%, corroborating the achievement of objective



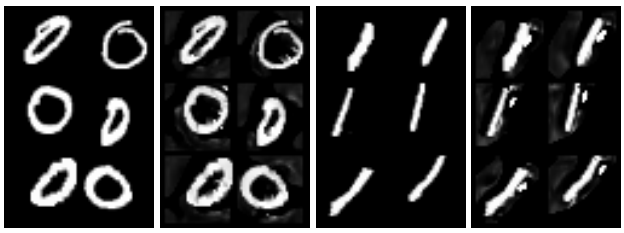
(a) 25% backdoor and 15% cross (b) 15% backdoor and 10% cross (c) 5% backdoor and 5% cross

Figure 2: Test accuracy on clean *test* images for **split-MNIST** under different poison ratios



(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 3: Test accuracy on clean *test* images for **split-MNIST** using a surrogate classifier with a different architecture



(a) Clean 0 (b) Poisoned 0 (c) Clean 1 (d) Poisoned 1

Figure 4: The clean and poisoned images used to train the replayer for **split-MNIST**.

O_{ste}. When the learner moves to the next task, the accuracy on the previous tasks falls significantly to around 25% despite some fluctuations. This confirms that the objective O_{eff} (forgetting) has also been attained.

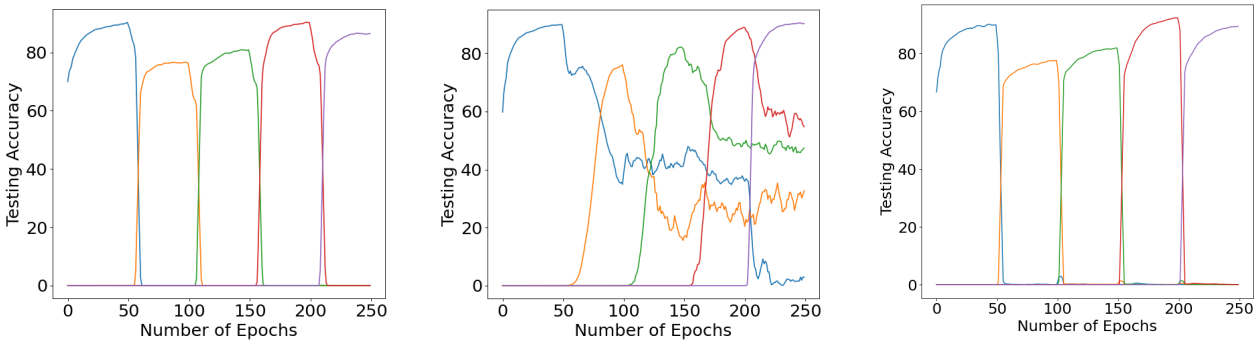
To study the impact of poison rate, we also tested with $\rho_b = 0.15$ and $\rho_c = 0.1$. As shown in Figure 2, the attack brings clean test accuracy on past tasks down to 30% (Figure 2b). Further reducing the poison ratio to $\rho_b = \rho_c = 0.05$ leaves the clean test accuracy above 70% (Figure 2c).

Additionally, we set up experiments on Split-MNIST where the surrogate classifier is different from the victim classifier in model architecture. We also used the reduced the poison ratio $\rho_b = \rho_c = 0.05$. As shown in Figure 3, test accuracy on past tasks has been reduced from 90% (Figure 3b) to 35% (Figure 3c), while the test accuracy on current tasks remains high. This further illustrated the proposed attack's effectiveness under lower poison ratio and without access to either the gradient or the architecture of the victim networks.

Finally, we plot in Figure 4 some example clean images of class 0 and 1 from the first task of split-MNIST, along with their corresponding poisoned images constructed by IAB (before label flipping). The poisons are quite inconspicuous.

split-CIFAR-10 To illustrate the effectiveness of CIAP in colored space, we repeated the experiment on CIFAR-10, with the same setup of five disjoint tasks. Here the victim model was trained for 50 epochs on each task.

Similar to the split-MNIST, the victim model completely forgets the earlier trained tasks after a new task starts, as shown in Figure 5a. After DGR is introduced in Figure 5b, although forgetting is not as well mitigated as in split-MNIST, solid improvement is still made on the test accuracy for all trained tasks. However, the improvements brought by DGR were completely eliminated by the CIAP attack. As Figure 5c shows, the test accuracy on past tasks drops to 0% after being poisoned. During the current task, however, the accuracy can still achieve the same level as in Figure 5a.



(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 5: Test accuracy on clean test images for **split-CIFAR-10**

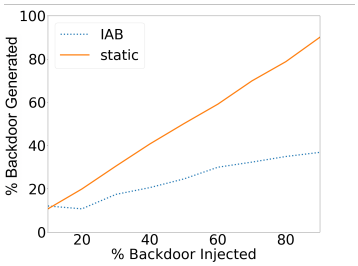
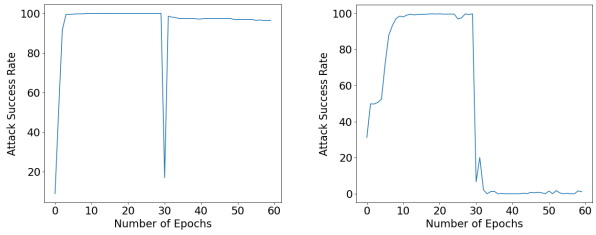


Figure 6: Poison rate of replayed images



(a) Static backdoor (b) Input-aware backdoor (IAB)

Figure 7: ASR for static backdoor and IAB

We also tested on the split-EMNIST dataset with 10 tasks, and the results are similar; see Appendix C. The results of FashionMNIST-MNIST and permuted-MNIST are in Appendix A and B, respectively.

5.2. Investigation of the trigger-discarding property

To better illustrate property $\mathbf{P}_{\text{dis}}$, we set up two experiments on split-MNIST using cWGAN. The **first** one studies the percentage of backdoored images (images with a trigger) generated by the replayer, when a varying portion of the training images are backdoored. Our goal is to show that such a percentage is much lower for IAB than for a static backdoor, i.e., the triggers of IAB are much less likely to survive the generative learning. A static backdoor refers to a white square on the image's top left corner. To this end, we trained a binary poison detector based on a training set that is backdoored with the IAB network learned from the first task. Another detector was trained analogously for static backdoor. This allows us to measure the percentage of backdoored images from the replayer. The two detectors achieve 97% accuracy, and similar ideas have been used in evaluating generative models such as inception score (Salimans et al., 2016).

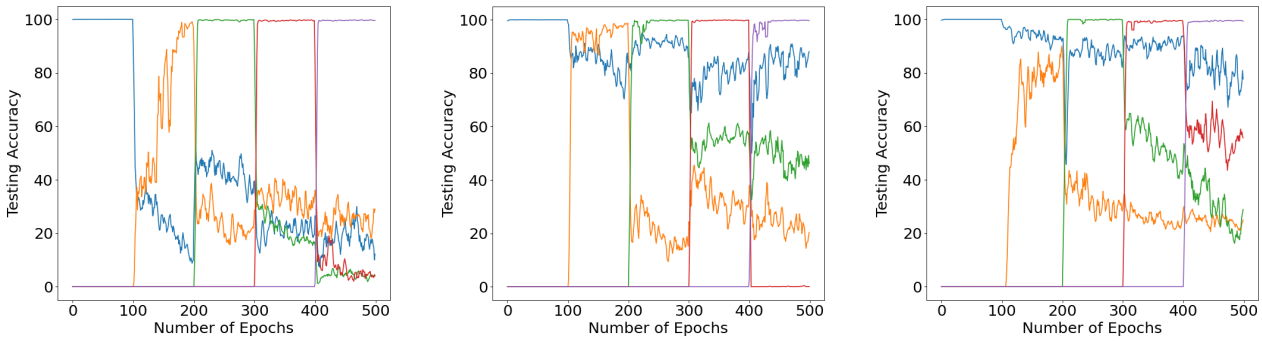
As shown in Figure 6, the static backdoor maintains almost the same percentage of backdoored images used for training, while that for IAB grows much more slowly, producing only 40% backdoored images when 90% of the training images are backdoored. This shows that IAB is far more likely to be discarded by the replayer.

Our **second** experiment examines the chance of replaying a backdoored image by using the attack success rate (ASR). In the same setting as the above experiment, we backdoored 25% images in the first 30 epochs with flipped labels, and used them to train a replayer. Then we generated examples from it in the second 30 epochs, and used them to incrementally train a new classifier. This classifier is tested on backdoored images, and the proportion of misclassified images is calculated as the ASR.

If the triggers are preserved by the replayer, then a classifier learned from the replayed images should permit a high ASR. This is confirmed in Figure 7a where the ASR remains at 100% for static backdoor. The drop in the middle is because the new classifier was trained from scratch. In contrast, the ASR drops to zero for IAB in Figure 7b, confirming that the newly trained *classifier* is not backdoored, i.e., the replayed images do not preserve triggers sufficiently well for training a backdoored classifier.

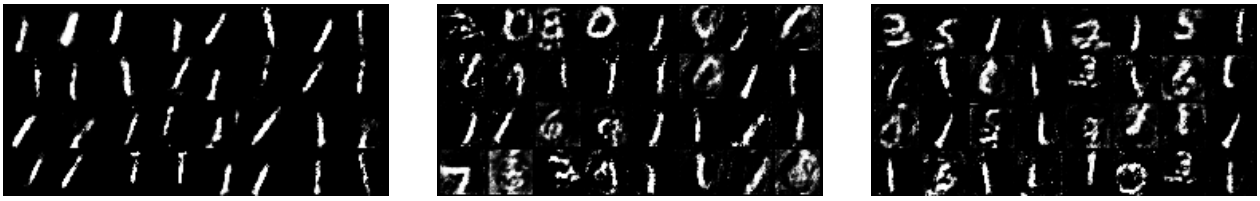
5.3. Assessing CIAP under defense (objective O_{rob})

We next study how well our attack withstands defenses. To this end, a ν-SVM with a radial basis kernel was applied to the output of the convolutional layer of SpinalVGG. This



(a) $\nu = 0.15$ (b) $\nu = 0.25$ (c) $\nu = 0.35$

Figure 8: Test accuracy after defense with different ν values for **split-MNIST**



(a) Clean replay (b) Poisoned replay (c) Poisoned replay after ν-SVM

Figure 9: Replayed images on **split-MNIST** with label "1" from (a) clean replayer, (b) poisoned replayer, and (c) poisoned replayer after filtered with ν-SVM ($\nu = 0.25$).

allows a portion of replayed samples to be filtered out, and the proportion is controlled by $\nu \in (0, 1)$.

As shown in Figure 8 where ν is varied in $\{0.15, 0.25, 0.35\}$ on the split-MNIST dataset, the filtering by ν-SVM does help a little, especially when ν is set around the poison ratio ($\rho_b = 0.25$). However, it remains unable to well remove the impact of the attack, and the test accuracy on past tasks still falls below 50%. Similar results on the other datasets are relegated to Appendix D.

The limited improvement could be partially ascribed to the compromised image quality due to the backdoors. To better visualize the consequence of poisoning on the replayer, we compare in Figure 9 the replayed images before and after the attack. Figure 9a presents example images generated by a replayer that is trained on clean images only. In contrast, Figure 9b shows that the poisoned replayer can often generate images from an incorrect class, i.e., another class that is incorrectly labeled as "1". Although the replayer cannot reproduce the input-aware backdoor, it tends to turn the backdoors into some random noise, making it harder for the filter to identify those poisons. As a result, the remaining replayed images are only slightly improved by the ν-SVM filtering as shown in Figure 9c.

We finally investigated the proportion of mislabeled example pairs generated by the replayer. The resulting confusion matrix is shown in Table 1 in Appendix E, using the replayers after completing task 5 on split-MNIST. The total "wrong pair ratio" turns out not high.

6. Conclusion and Future Work

We proposed a novel poisoning attack on the generative replayer in continual learning, so that forgetting can be promoted while the accuracy at the current task is not hurt. Our approach takes advantage of input-aware backdoor attacks, whose triggers cannot be well captured by normal generative models thanks to their input dependency. In future work, we will delve more into the theoretical analysis of the trigger-discarding property. We will also extend the approach to continual learning without known task boundaries.

Societal impact. We revealed important vulnerabilities in continual learning methods that are based on generative rehearsal. Similar to many works that identify an attack without effective countermeasures available in the existing literature, we will address this defense issue in the future.

Acknowledgements

We thank the reviewers for their constructive comments. This work is supported by NSF grant RI:1910146.

References

Biggio, B., Nelson, B., and Laskov, P. Poisoning attacks against support vector machines. In *International Conference on Machine Learning (ICML)*, 2012.

Burkard, C. and Lagesse, B. Analysis of causative attacks against svms learning from data streams. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*, pp. 31–36, 2017.

Chen, D., Yu, N., Zhang, Y., and Fritz, M. GAN-leaks: A taxonomy of membership inference attacks against generative models. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

Chen, X., Liu, C., Li, B., Lu, K., and Song, D. Targeted backdoor attacks on deep learning systems using data poisoning. *ArXiv*, abs/1712.05526, 2017.

Chen, Z. and Liu, B. *Lifelong Machine Learning: Second Edition*. Morgan & Claypool Publishers, 2018.

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. High-performance neural networks for visual object classification. *CoRR*, abs/1102.0183, 2011. URL http://arxiv.org/abs/1102.0183.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. URL http://arxiv.org/abs/1702.05373.

Condessa, F. and Kolter, Z. Provably robust deep generative models. *ArXiv*, abs/2004.10608, 2020.

Cong, Y., Zhao, M., Li, J., Wang, S., and Carin, L. GAN memory with no forgetting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Ding, S., Tian, Y., Xu, F., Li, Q., and Zhong, S. Poisoning attack on deep generative models in autonomous driving. In *EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2019.

Engelmann, J. and Lessmann, S. Conditional wasserstein gan-based oversampling of tabular data for imbalanced learning. *Expert Systems with Applications*, 174:114582, 2021.

Geiping, J., Fowl, L. H., Huang, W. R., Czaja, W., Taylor, G., Moeller, M., and Goldstein, T. Witches' brew: Industrial scale data poisoning via gradient matching. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=01olnfLIbD.

Goldblum, M., Tsipras, D., Xie, C., Chen, X., Schwarzschild, A., Song, D., Madry, A., Li, B., and Goldstein, T. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (02):1563–1580, 2023.

Gong, Y., Li, B., Poellabauer, C., and Shi, Y. Real-time adversarial attacks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Goodfellow, I. J., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. An empirical investigation of catastrophic forgeting in gradient-based neural networks. In Bengio, Y. and LeCun, Y. (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6211.

Gu, T., Dolan-Gavitt, B., and Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *ArXiv*, abs/1708.06733, 2019.

Hayes, J., Melis, L., Danezis, G., and Cristofaro, E. D. Logan: Membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies*, 2019:133–152, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hilprecht, B., Härterich, M., and Bernau, D. Monte carlo and reconstruction membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies*, 2019(4):232–249, 2019.

Hu, H. and Pang, J. Model extraction and defenses on generative adversarial networks. *arXiv preprint arXiv:2101.02069*, 2021.

Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., and Li, B. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

Kabir, H. M. D., Abdar, M., Jalali, S. M. J., Khosravi, A., Atiya, A. F., Nahavandi, S., and Srinivasan, D. Spinalnet: Deep neural network with gradual input. *CoRR*, abs/2007.03347, 2020. URL https://arxiv.org/abs/2007.03347.

Kos, J., Fischer, I., and Song, D. Adversarial examples for generative models. *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 36–42, 2018.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

Levine, A. and Feizi, S. Deep partition aggregation: Provable defense against general poisoning attacks. In *International Conference on Learning Representations (ICLR)*, 2021.

Lin, Y.-C., Hong, Z.-W., Liao, Y.-H., Shih, M.-L., Liu, M.-Y., and Sun, M. Tactics of adversarial attack on deep reinforcement learning agents. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Liu, Y., Ma, S., Aafer, Y., Lee, W., Zhai, J., Wang, W., and Zhang, X. Trojaning attack on neural networks. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

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

Mladenovic, A., Bose, A. J., Berard, H., Hamilton, W. L., Lacoste-Julien, S., Vincent, P., and Gidel, G. Online adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2022.

Murphy, K. P. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL https://probml.github.io/pml-book/book2.html.

Muñoz-González, L., Pfitzner, B., Russo, M., Carnerero-Cano, J., and Lupu, E. C. Poisoning attacks with generative adversarial nets. *arXiv preprint arXiv:1906.07773*, 2019.

Nguyen, A. and Tran, A. Input-aware dynamic backdoor attack. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Online Supplementary. Supplementary material including code (no tracking). https://www.dropbox.com/sh/mku8oln1t7ngscl/AABVPSwZBlx41GtQYRyYVRgha?dl=0.

Peri, N., Gupta, N., Huang, W., Fowl, L., Zhu, C., Feizi, S., Goldstein, T., and Dickerson, J. P. Deep k-NN defense against clean-label data poisoning attacks. *arXiv: Learning*, 2019.

Salem, A., Sautter, Y., Backes, M., Humbert, M., and Zhang, Y. BAAAN: Backdoor attacks against autoencoder and gan-based machine learning models. *arXiv preprint arXiv:2010.03007*, 2020.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Shafahi, A., Huang, W., Najibi, M., Suciu, O., Studer, C., Dumitras, T., and Goldstein, T. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Sohn, K., Yan, X., and Lee, H. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Stadler, T., Oprisanu, B., and Troncoso, C. Synthetic data – anonymisation groundhog day. *arXiv preprint arXiv:2011.07018*, 2022.

Steinhardt, J., Koh, P. W. W., and Liang, P. S. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3517–3529, 2017.

Sun, J., Zhang, T., Xie, X., Ma, L., Zheng, Y., Chen, K., and Liu, Y. Stealthy and efficient adversarial attacks against deep reinforcement learning. In *National Conference of Artificial Intelligence (AAAI)*, 2020.

Tabacof, P., Tavares, J., and Valle, E. Adversarial images for variational autoencoders. *ArXiv*, abs/1612.00155, 2016.

Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723, 2019.

Wang, Y. and Chaudhuri, K. Data poisoning attacks against online learning. *arXiv preprint arXiv:1808.08994*, 2018.

Xiao, H., Xiao, H., and Eckert, C. Adversarial label flips attack on support vector machines. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pp. 870–875, 2012.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Yang, C., Wu, Q., Li, H., and Chen, Y. Generative poisoning attack method against neural networks. *ArXiv*, abs/1703.01340, 2017.

Zhang, X., Zhu, X., and Lessard, L. Online data poisoning attack. In *Conference on Learning for Dynamics and Control*, 2020.

Supplementary Material

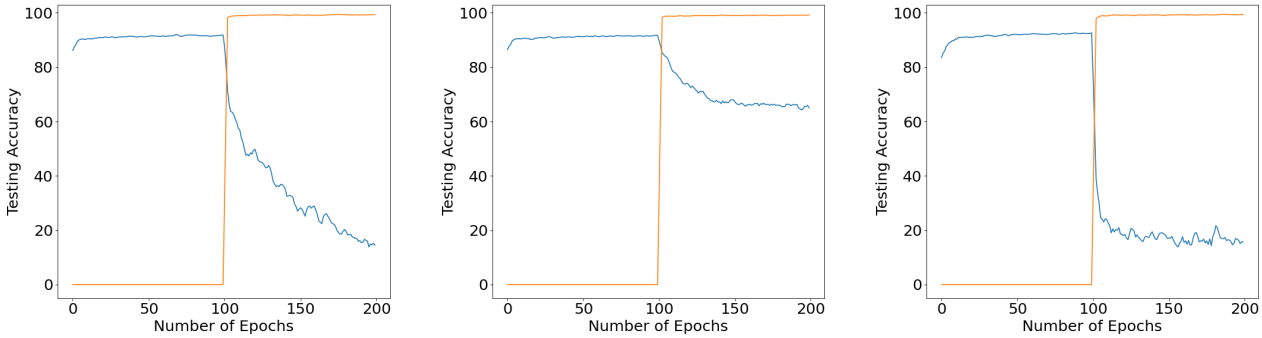
A. Results of FashionMNIST-MNIST on cWGAN

There are two tasks in this task-incremental experiment: FashionMNIST followed by MNIST. Both tasks have 10 classes, and the victim model's goal is to classify images from them into a *shared* set of labels – "T-shirt" and "hand written 0" are both associated with "class 0". The victim model was trained for 100 epochs with samples from the first task, followed by 100 epochs on the second task. Since the two tasks share the same label space, the victim model will still achieve 10% testing accuracy on the past task even after its been completely forgotten.

Figure 10a shows the baseline result without a replayer, where the blue and orange lines represent the test accuracy of the first and second tasks, respectively. As expected, the test accuracy for FashionMNIST declines rapidly from 92% to 14% as the training on MNIST progresses to 100 epochs.

Figure 10b shows the result of DGR-facilitated training. Although the test accuracy of FashionMNIST still drops a bit after MNIST starts to be trained on, it stabilizes at 66%, which illustrates the effectiveness of DGR and the sufficient capacity of the cWGAN.

Figure 10c shows the result after our attack is introduced. In the first task (epochs 0-100), the test accuracy remains very similar to that of Figure 10a and 10b, corroborating the achievement of objective O_{ste}. When the continual learning moves on to the second task, the test accuracy on the earlier task falls significantly to 17%, much lower than the 93% achieved at the end of the first task. This confirms that the objective O_{eff} (forgetting) has also been attained.

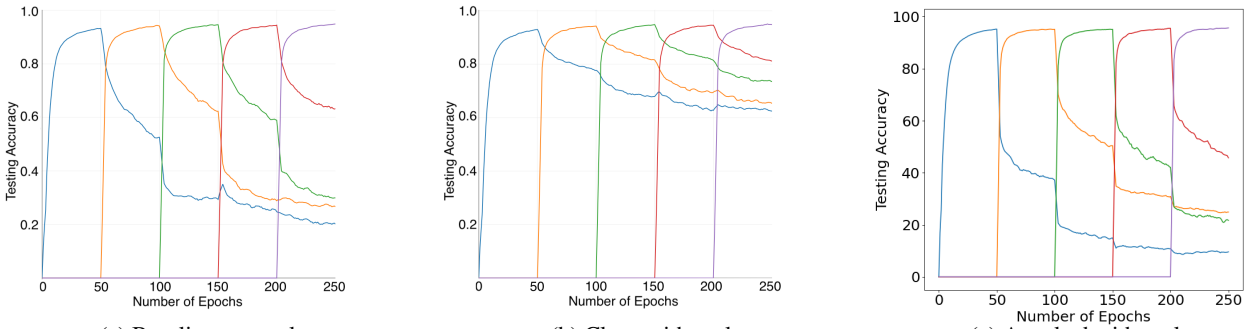


(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 10: Test accuracy on clean testing images for **FashionMNIST-MNIST**

B. Results of permuted-MNIST on cWGAN

In this dataset, each task consists of images from all the 10 classes of MNIST. However, each task also employs a unique pixel-level permutation, applied to all the images. The performance is similar to split-MNIST. In Figure 11a where no replayer is used, the test accuracy drops to 20% after new tasks start. Similar to FashionMNIST-MNIST in Appendix A, since all the tasks in permuted-MNIST share the same label space, even random guessing would give 10% accuracy. So this 20% is already very close to complete forgetting. Replay ameliorated the problem, but the gain is much obliterated by the CIAP attack in Figure 11c.



(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

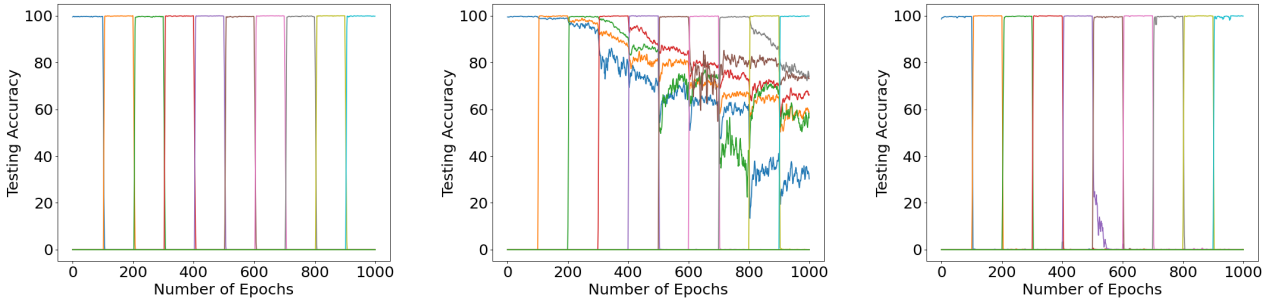
Figure 11: Test accuracy on clean test images for **permuted-MNIST**

C. Results of split-EMNIST on cWGAN

Our experiment setup largely follows Shin et al. (2017). Although many continual learning literature also evaluates on 5 tasks or fewer, it is of interest to study more tasks. Note that there are only 10 classes in the split-MNIST dataset, and we have already used up all the classes in the five tasks. Although additional tasks can be constructed by grouping new pairs of classes, it will not serve our investigation well because the sixth task would then include two classes that have already been witnessed before. Although our attacker has promoted catastrophic forgetting, it is still much easier than starting from scratch like in tasks 1-5, i.e., exhibiting no struggle.

As such, we developed a new experiment with the EMNIST dataset (balanced split), which consists of handwritten digits and letters in 47 classes. Since some classes looks alike (i.e. "C" and "c", "S" and "s"), Cohen et al. (2017) merged those similar classes, and balanced the merged classes. We constructed 10 disjoint binary tasks out of it, and the test accuracy on clean test images is shown in Figure 12.

Clearly, the replayer is helpful in alleviating catastrophic forgetting (Figure 12b), and our attacker is able to exacerbate forgetting in all tasks by reducing the test accuracy (Figure 12b). In particular, with the poisoned replayers, all tasks achieve very high accuracy during its own training (Figure 12b). It is noteworthy that in this dataset the help of replayer is not sufficient enough to completely eliminate forgetting. This, however, does not diminish our contribution as an attacker, because the attacker only needs to further impair the accuracy on top of the performance achieved by the replayer.

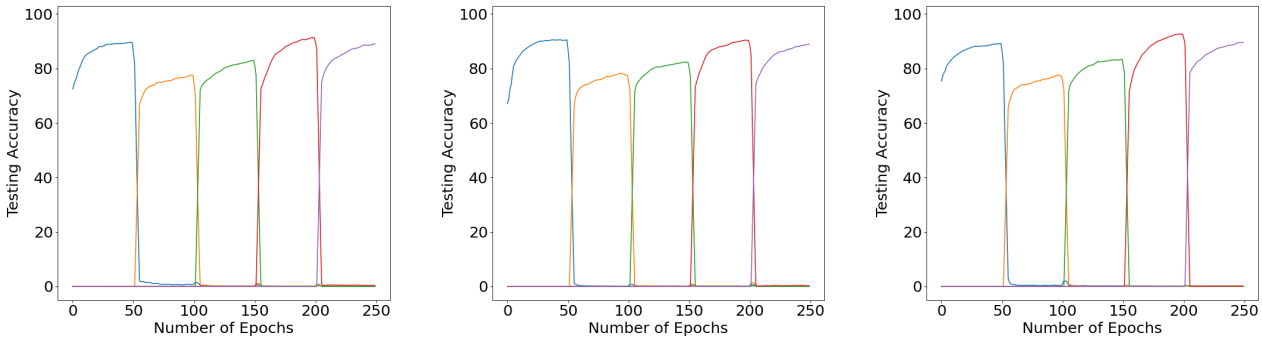


(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 12: Test accuracy on clean test images for **split-EMNIST**

D. Results of ν-SVM defense on split-CIFAR-10

Similar to the experiment on split-MNIST in Section 5.3, we tested the attack with ν-SVM defense on CIFAR-10. As shown in Figure 13, with three different values of ν, the test accuracy on past tasks dropped to almost 0 after switching to a new task. This confirms that our CIAP remains effective under the defense of ν-SVM.



(a) $\nu = 0.15$ (b) $\nu = 0.25$ (c) $\nu = 0.35$

Figure 13: Test accuracy after defense with different ν values for **split-CIFAR-10**

E. Proportion of mislabeled pairs in replayed data

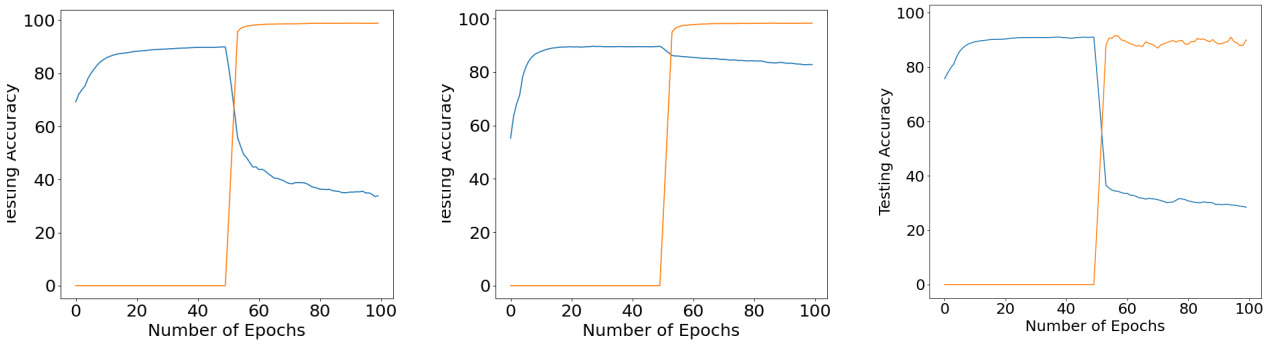
Table 1: Proportion of mislabeled pairs in replayed data

	0	1	2	3	4	5	6	7	8	9
0	**6.25**	0.15	0.89	0.18	0.61	0.23	0.7	0.09	0.21	0.63
1	0.01	**8.19**	0.31	0.02	0.1	0.08	0.01	0.49	0.63	0.1
2	0.03	0.14	**6.26**	0.98	0.07	0.67	0.39	0.42	0.62	0.36
3	0.01	0.08	0.65	**7.45**	0.74	0.1	0.18	0.26	0.12	0.35
4	0.02	0.09	0.05	0.04	**7.49**	0.46	0.22	0.53	0.44	0.62
5	0.03	0.04	0.03	0.19	0.48	**7.09**	1.21	0.04	0.46	0.39
6	0.1	0.11	0.07	0.04	0.04	0.23	**8.92**	0.11	0.06	0.27
7	0.13	0.36	0.2	0.09	0.13	0.16	0.02	**7.42**	0.21	1.22
8	0.01	0.01	0.0	0.0	0.3	0.0	0.04	0.01	**9.08**	0.49
9	0.0	0.0	0.0	0.04	0.15	0.11	0.01	0.01	1.62	**8.02**

Since the poisoned generator itself does not provide a flag indicating whether a generated feature-label pair is wrong, we trained a classifier C^* on clean MNIST data and applied it to the generated data pairs. The resulting confusion matrix is shown in Table 1 in percentage, using the replayers after completing task 5 on split-MNIST. The columns are the labels produced by C^*, while the rows are the labels used to invoke the generator, i.e., used as the label for the replayed data. The total "wrong pair ratio" is 24% (sum of off-diagonal values), which is not that high.

F. Results of FashionMNIST-MNIST on cVAE

Similar to the experiment with cWGAN in Appendix A, we tested the attack with cVAE replayer on FashionMNIST-MNIST. As shown in Figure 14, the testing accuracy on the earlier task dropped from 91% to 27% after attack. This indicates that cVAE is also vulnerable to the proposed attack.

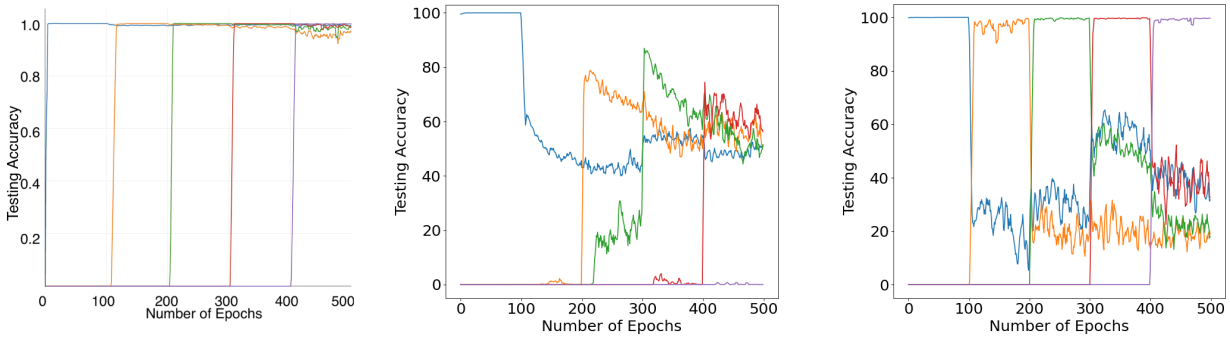


(a) Baseline: no replayer (b) Clean with cVAE replay (c) Attacked with cVAE replay

Figure 14: Test accuracy on clean test images for **FashionMNIST-MNIST** based on cVAE

G. Baseline attack on Split-MNIST

To better illustrate the effectiveness of our attack, we conducted a baseline naive attack on Split-MNIST by flipping 25% labels without changing the corresponding images. As shown in Figure 15b, starting from the second task (orange line), the test accuracy of current tasks drops to below 20%, compared with 100% without attack (Figure 15a). Thus the baseline attack will be discerned by the victim during training of the current task, violating objective O_{ste}.



(a) Baseline performance: clean with replay. Same as Figure 1b.

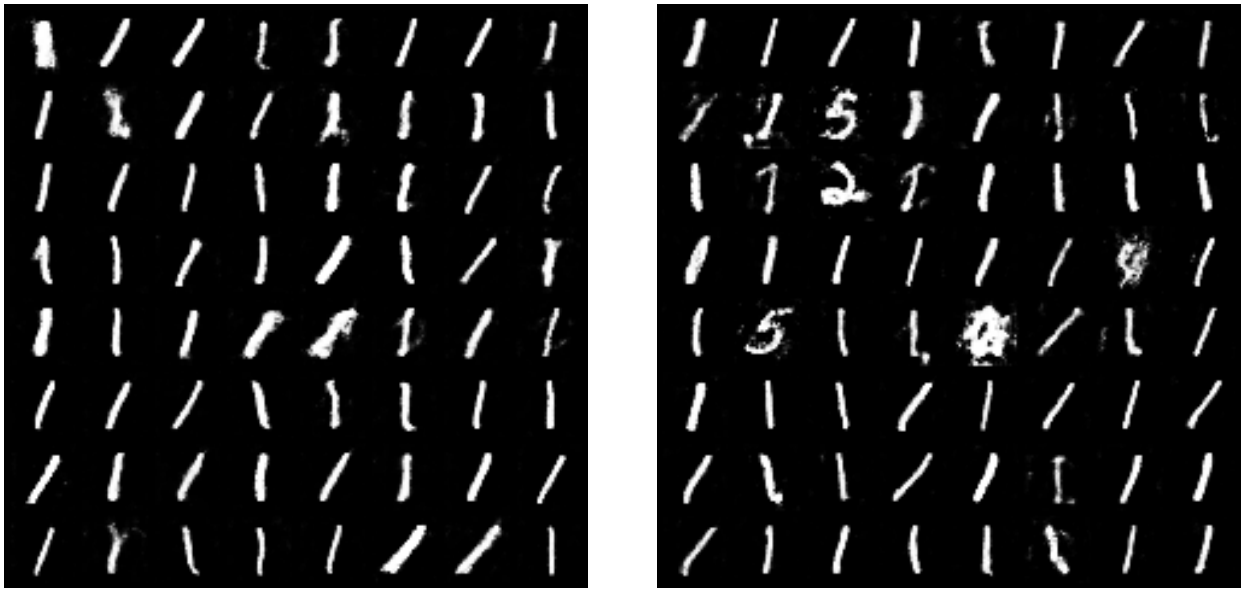
(b) Baseline attack with simple label flipping: naive attack with replay

(c) CIAP attack with replay. Same as Figure 1c.

Figure 15: Test accuracy on clean test images for **Split-MNIST**

H. Replayed images on Split-MNIST

Following the experiment in Appendix **??**, we also printed out examples of replayed images with label "1" before and after the attack in Figure 16. With reduced poison ratio, the poisoned replay in Figure 16b contains less images from other classes than it was in Figure 9b.



(a) Clean replay

(b) Poisoned replay

Figure 16: Replayed images on **split-MNIST** with label "1" from (a) clean replayer, (b) poisoned replayer