Continual Poisoning of Generative Models to Promote Catastrophic Forgetting

Siteng Kang, **Zhan Shi,** **Xinhua Zhang**
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607
{skang98, zshi22, zhangx}@uic.edu

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 backdoor 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 [1], flip the training labels [2, 3], or modify the training example inputs [4, 5].

Although poisoning attacks have been extensively studied under discriminative learning, their potential risk in *generative* learning has been largely understudied. [6] 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. [7] 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 batch.

The increasing penetration of generative models in machine learning urges investigation of poisoning attacks on it 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 [8]. 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 [9].

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 [10, 11]. 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:

O1 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.

O2 *During* task t, the classifier trained from the poisoned data does *not* suffer degradation of test accuracy on task t itself. This is important because poor performance on the current task can

ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

raise significant and 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 the access to the samples of task t is already lost.

O3 The poison should sustain solid defense deployed by both the classifier and the replayer.

The main difficulty lies in two folds. Firstly, although both **O1** and **O2** are straightforward to fulfill individually, they are at odds with each other and are hard to fulfill simultaneously. Secondly, the transiency of data stream compels the adversary to make irrevocable attacks at task t before future tasks arrive and before catastrophic forgetting can start to occur. Due to this difficulty, poisoning attacks have been much less studied in an online setting. [12] addressed the online decision problem of selecting k examples for evasion attack. [13] assumed the instances are drawn i.i.d. from a time-invariant distribution, which is not the case in continual learning because tasks may even have disjoint classes. Other works require multiple passes of the data stream [14–16], or clairvoyant knowledge of future data [17, 18]. Appendix A further discussed our contribution and related works.

2 Attacking 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:

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

Here ℓ is a loss function, and we will focus on multi-class classification with cross-entropy loss. A major challenge in continual learning is that at task t, only the samples from the current P_t are available, while those from the previous tasks are no longer accessible. Although most algorithms would only adapt to the new task instead of completely retraining from scratch, the performance on previous tasks may still deteriorate significantly, a phenomenon known as *catastrophic forgetting*.

To alleviate this issue, DGR-based approaches resort to learning a DGR model G_t that approximately replicates P_t. Due to the constraints in computation and storage, it has to be a lossy approximation because otherwise one might as well store all the past examples. 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.

The DRG models can be simplified into a **single** replayer G that is updated over time, as opposed to one replayer per task. However, our contribution is the **poisoner**, not the replayer. Employing multiple replayers only makes attacks even more challenging, because instead of just poisoning one running replayer, we are now missioned to poison many of them, each of which can only be poisoned once at its current task. If we simply keep a single running replayer, then we enjoy many opportunities of poisoning it at any time and our objectives will become much easier to achieve. To conclude, multiple replayers set up 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 the cGAN.

2.1 Attackers and Learners

Two parties participate in the process, and we first set forth our assumptions on them. 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. We also assume that the learner cannot store any data beyond its current task, which is standard in continual learning.

A defender is an algorithm that the learner employs 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, whose counterpart—*post-training* defense—patches up the learned model [19].

The **attacker** (our threat model) is *only allowed to poison (modify) the samples $\mathcal{D}_t$ in step 5 of Algorithm 1*. The attacker has no access to the internal mechanism of classifier, replayer, or defender. It can *read* the gradient of the classifier's training objective with respect to the input, but not its structure or weights. This is a moderate mid-ground between full access (e.g., training by the attacker itself on the cloud) and no access (independently manipulating the training examples). Such an assumption has been commonly adopted, e.g., by dynamic backdoors [20], Witches' brew [21], and by implicit differentiation based poisoning [22]. 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 Appendix C).

Algorithm 1: Deep generative replay (DGR) used by continual learning to combat catastrophic forgetting

```
  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 $P_t$          $\leftarrow$ 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
```

Achieving poisoning through backdoor. Although our method will leverage backdoor attacks, our overall goal is poisoning attack, *not* backdoor. In backdoor attacks, a classifier predicts poorly only on backdoored examples with triggers, while remaining well on clean test examples. In contrast, poisoning attacks (excluding backdoor) aim to predict poorly on *all* examples, irrelevant of "trigger".

In continual learning, an attacker generally does not have the liberty of planting a trigger on test examples. So we will address in **O1** and **O2** a much more challenging setting (from an attacker's perspective) where such an access is not available. It is important to note that our approach only utilizes backdoor attacks as a means of achieving the goal of poisoning the generator/replayer. Further discussion on the difficulties of attack is in Appendix B.

3 The Attack Algorithm

Our poisoning attack proposes evading the defense by leveraging the *input-aware backdoor* attack [23], so that *mislabeled* data points carrying a *trigger* can be injected to $\mathcal{D}_t$ in a small amount. In particular, our approach achieves **O1** to **O3** 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.
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. Such a flexibility of modulation proves essential. Backdoor attacks such as BadNets [24] 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:

P1 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.

P2 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 **P2**).

This is indeed challenging as we experimented. Static backdoor (BadNet) can be easily replayed by a generative model. Trojan attack requires access to the victim model's structure and weights. Neither can it survive the defense of neural cleansing. We also tested static backdoor with changing location, which again, turned out easily detected and fixed by neural cleansing. Witches' Brew [21] 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. Please refer Appendix C for more detail.

According to [23] and also observed in our experiment, IAB proffers the following property:

P3 The backdoor in IAB can be hardly detected by state-of-the-art methods, e.g., neural cleansing.

To summarize, we fulfilled **O2** by **P2**, and **O3** by **P3**. To meet **O1**, we require a *trigger-discarding* property as follows, which plays a key role in our method and will be discussed in the next subsection:

P4 The replayer cannot well capture the trigger generation network of IAB, in the sense that the replayed examples do *not* well preserve the triggers.

3.2 Trigger-discarding generative models

Property **P4** depends on both the replayer and the trigger. If the trigger is a constant small white square at the image center, most generative models will preserve it. 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 and $\mathbb{E}_{x_i \sim Q_x^\theta} \mathcal{A}(\{x_i\}_i)$.

Directly computing the gradient in θ is both expensive and infeasible, because $\mathcal{A}$ is assumed inaccessible in Section 2.1. Fortunately, our experiments show that **P4** is well (but not perfectly) achieved when IAB is applied in conjunction with several SOTA generative models such as conditional Wasserstein GAN [cWGAN, 25] and conditional VAE [26]. This is no surprise because these models have limited capacity, and the trigger's dependency on the input, which is more involved than just random, significantly raises the sample complexity for generative learning. A theoretical analysis is left for future work, and Appendix F.2 empirically illustrates this intriguing property.

4 Poisoning Replayer via IAB

We are now ready to apply IAB to poison the DGR used by continual learning against forgetting. We will call our method Continual Input-Aware Poisoning (CIAP). It does *not* backdoor test images. Details of CIAP can be found in Algorithm 4 in Appendix D. We conducted experiments on five continual learning datasets to verify those aforementioned objectives and proprieties of CIAP in Appendix F. Conclusion is made in Appendix G.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[20] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang. Dynamic backdoor attacks against machine learning models. *arXiv preprint arXiv:2003.03675*, 2020.

[21] J. Geiping, L. H. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein. 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`.

[22] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017.

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

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

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

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

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

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

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

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

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

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

[33] D. Chen, N. Yu, Y. Zhang, and M. Fritz. 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.

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

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

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

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

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

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

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

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

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

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

[44] I. J. Goodfellow, M. Mirza, X. Da, A. C. Courville, and Y. Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *In* Y. Bengio and Y. LeCun, 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`.

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

[46] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Tech. Rep. 0*, University of Toronto, Toronto, Ontario, 2009.

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

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

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

Supplementary Material

A Our contribution and Related work

Our contribution, therefore, is to overcome these challenges and to reveal the vulnerability of generative models in the sense that their training data can be poisoned stealthily such that a task can be learned well at present but forgotten soon in the future. Noting that simple label-flipping poisoning can be easily detected, we resort to dirty-label backdoor/Trojan attack [27] to attain **O2**: the trained classifier performs correctly on clean examples, but errs if the example is planted with a trigger. To further achieve **O1** and **O3**, we capitalize on the input-aware backdoor [23], which allows the trigger to vary depending on the image. As a result, it can not only withstand stronger defense (§E), 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 §3.2) while retaining the incorrect label, leading naturally to forgetting (§3). The problem is set up in §2, and experiments are provided in §F 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 particularly 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 [28, 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, [29] learned robust variational autoencoder (VAEs) that retain high likelihood for the data points under adversarial perturbation. The underlying threat is evasion attack, and along similar lines, [30] and [31] studied attacks that promote reconstruction error of the decoder in a VAE. Some recent works address attacks on membership inference [32–34], model extraction [35], and attribute inference [36]. 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 [37, 38]. 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 [3, 39–41].

B 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 simply flipping the label of some examples in $\mathcal{D}_t$, because it is easy to detect [3] and the resulting classifier C_t will perform poorly on task t.

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.

C Details about input-aware backdoor

We first recap the input-aware backdoor attack [IAB, 23] as shown in Algorithm 2 under a given data distribution P, and then detail how it can be utilized for our purpose. In line 2, 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 can contribute in one of the following three 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 5 via elementwise product $\odot$.

- As a cross-trigger example with probability ρ_c. To ensure that a trigger synthesized for one example is *not* effective for another, another $\hat{x}$ is sampled from P with label $\hat{y}$. Then x is injected with the trigger generated from $\hat{x}$ for a wrong label y', and the result is paired with the clean label y (line 6).
- As a clean example otherwise (line 7).

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$(P, B_{[\mathcal{Y}]})$
2 return $\arg\min_{C, B_{[\mathcal{Y}]}} \{\mathcal{L}_{cl} + \lambda_{div}\mathcal{L}_{div}\}$, where $\mathcal{L}_{cl} = \sum_{(x,y)\in\tilde{\mathcal{D}}} \ell(C(x), y)$ is the classif. risk
```

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 $(\hat{x}, \hat{y})$ from $P$ excluding $(x, y)$,   sample $y'$ from $\mathcal{Y} \backslash \{y\}$,
4     $\mathcal{L}_{div}$ += $\|x - \hat{x}\| \,/\, \|B_{y'}(x) - B_{y'}(\hat{x})\|$  // Accrue the diversity loss
5     if $d < \rho_b$ then  $x' \leftarrow x \odot B_{y'}(x)$,  $\tilde{\mathcal{D}}$ += $(x', y')$  // make a backdoor example
6     else if $d < \rho_b + \rho_c$ then  $x' \leftarrow x \odot B_{y'}(\hat{x})$,  $\tilde{\mathcal{D}}$ += $(x', y)$  // make a cross example
7     else  $\tilde{\mathcal{D}}$ += $(x, y)$  // clean example
8 return $\tilde{\mathcal{D}}$ and $\mathcal{L}_{div}$
```

In Algorithm 2, $B_{[\mathcal{Y}]}$ 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 $B_{[\mathcal{Y}]}$, which is then optimized in line 2 along with the classifier C. To see that the attacker fits in our threat model, note C and B are jointly optimized in line 2 of Algorithm 2, and the attacker only requires *reading* the gradient with respect to the input of C.

D Detailed CIAP Algorithm

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 $t$
   Input: Classifier $C_{t-1}$, and backdoor generators $B_{[\mathcal{Y}]}$
 1 Initialize $C$ with $C_{t-1}$.
 2 for $i \in [t - 1]$ do
 3     $S_i \leftarrow$ SampleFromReplayer$(G_i)$ ($X, Y$ pairs)
 4 Defender: Apply $\nu$-SVM on the replayed data $S_{1:t-1}$
 5 for number of iteration (run in mini-batches) do
 6     $\tilde{\mathcal{D}}_t, \mathcal{L}_{div} \leftarrow$ InputAwareBackdoor–Obj$(P_t, B_{[\mathcal{Y}]})$ // Both $\tilde{\mathcal{D}}_t$ and $\mathcal{L}_{div}$ are functions of $B_{[\mathcal{Y}]}$
 7     $\mathcal{L}_{cl} \leftarrow$ sum of $\ell(C(x), y)$ over $(x, y) \in \tilde{\mathcal{D}}_t \cup S_{1:t-1}$ // $\mathcal{L}_{cl}$ is a function of $C$ and $B_{[\mathcal{Y}]}$
 8     User: update $C$ to reduce $\mathcal{L}_{cl}$
 9     Attacker: update $B_{[\mathcal{Y}]}$ to reduce $\mathcal{L}_{cl} + \lambda_{div}\mathcal{L}_{div}$
10     Update the replayer $G_t$ using $\tilde{\mathcal{D}}_t$ and the latest $B_{[\mathcal{Y}]}$
11 Defender: Neural Cleansing on $C_t \leftarrow C$
12 return $G_t, B_{[\mathcal{Y}]}$, and $C_t$
```

Algorithm 4 demonstrates the operation of all participants (user, attacker, and defender) during task t, which corresponds to the loop under a given t in Algorithm 1 (line 3 to 7 therein). The red colored steps are reserved for defense, which will be detailed in Appendix E.

In line 7, the poisoned data $\tilde{\mathcal{D}}_t$ for task t is joined with the replayed data $S_{1:t-1}$ to construct the classification risk $\mathcal{L}_{cl}$. Both the attacker and the user are trained in the same way as in Algorithm 2, while the only difference is that our optimization here is based on mini-batches, and the replayer G_t is additionally learned in line 10.

E The Defense Algorithm

We consider two defenses against the CIAP attack. The first is neural cleansing [19], which has been inserted in line 11 of Algorithm 4. If it were successful, then the backdoor planted in C_t would be detected and removed, thereby defeating objective **O2** with immediate poor accuracy on clean test examples at task t.

Our second defense is aimed at objective **O1**. To this end, we apply an outlier detector ν-SVM to $S_{1:t-1}$, which is in line 4 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 we did poison the replayer 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 add an additional source of complication and confusion to the generative model. Indeed, we experimented with directly generating label-flipped examples based on clean images, and ν-SVM easily filtered them out. However, as shown in Appendix E, this is not the case when ν-SVM is applied to our replayed images.

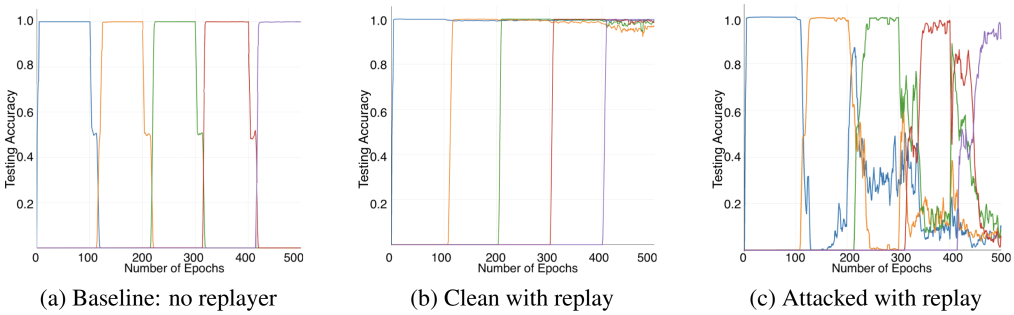
F Experiments

We conducted experiments on CIAP and verified: i) it attains the two objectives **O1** and **O2**, details in Appendix F.1; ii) the trigger-discarding property introduced in Section 3.2 holds true for commonly used generative models, details in Appendix F.2; iii) CIAP remains effective under strong defenders (**O3**), details in Appendix E. The code is available at [42].

F.1 Effectiveness of the attack for objectives O1 and O2

We tested CIAP on four datasets: Split-MNIST [43], Permuted-MNIST [44], FashionMNIST-MNIST [45], and Split-CIFAR-10 [46]. We used SpinalVGG as the victim classifier [47] for the three MNIST datasets, and ResNet [48] for the Split-CIFAR-10 dataset. The results shown here use cWGAN with gradient penalty as the replayer. The poison ratio was set to $\rho_b = 0.25$, and the cross ratio to $\rho_c = 0.15$. Each trigger was allowed to change 10% of the pixels of a selected image (mask density), and Figure 2 will show that the triggers are quite inconspicuous.

Split-MNIST We split 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.



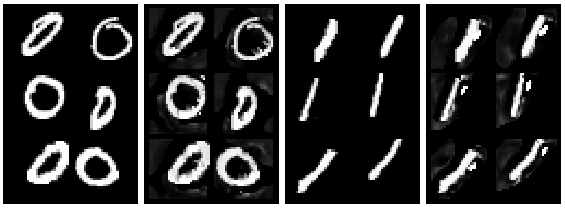
(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 1: Test accuracy on clean testing images for **Split-MNIST**

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 **O2**. When the learner moves to the next task, the accuracy on the previous tasks falls significantly to around 20% despite some fluctuations. This confirms that the objective **O1** (forgetting) has also been attained.

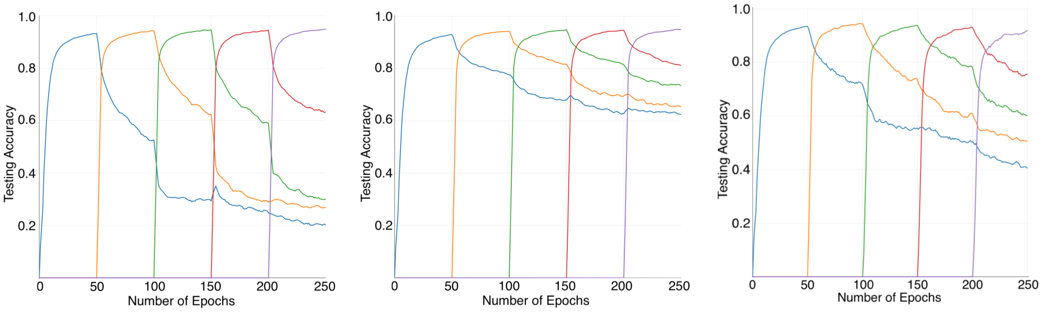
Finally, we plot in Figure 2 some example clean images of class 0 and 1 from the first task of Split-MNIST, along with their corresponding poisoned images constructed by IAB (before label flipping). Clearly the poisons are quite inconspicuous.

Permuted-MNIST Here, 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.



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

Figure 2: The clean and poisoned images used to train the replayer for **Split-MNIST**.

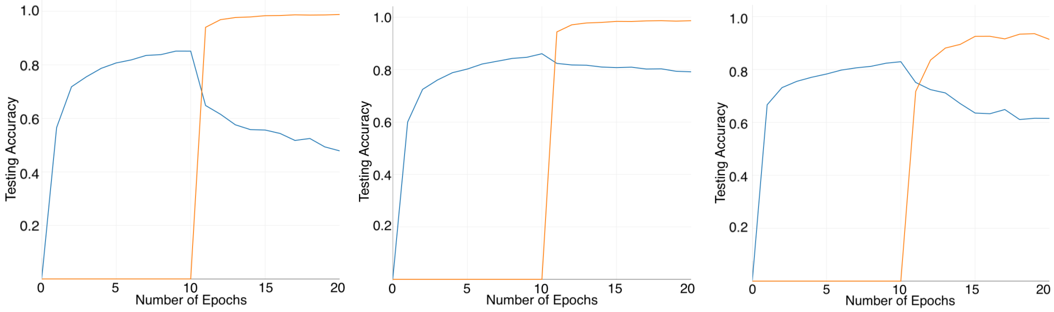


(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 3: Test accuracy on clean testing images for **Permuted-MNIST**

In Figure 3a where no replayer is used, the test accuracy drops to 20% after new tasks start. 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 as in Figure 3b, but the gain is much obliterated by the CIAP attack in Figure 3c.

FashionMNIST-MNIST There are two tasks in this dataset: 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 10 epochs with samples from the first task, followed by 10 epochs on the second task.

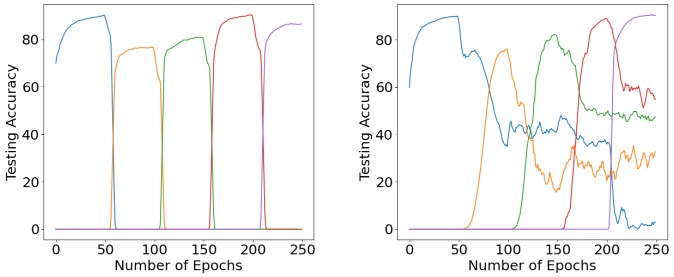


(a) Baseline: no replayer (b) Clean with replay (c) Attacked with replay

Figure 4: Test accuracy on clean testing images for **FashionMNIST-MNIST**

Figure 4a 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 85% to 48% as the training on MNIST progresses to 10 epochs. Figure 4b 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 79%, which illustrates the effectiveness of DGR and the sufficient capacity of the cWGAN. Figure 4c shows the result after our attack is introduced. In the first task (epochs 0-10), the test accuracy remains very similar to that of Figure 4a and 4b, corroborating the achievement of objective **O2**. When the continual learning moves on to the second task, the test accuracy on the earlier task falls significantly to 61%, much lower than the 79% achieved at the end of the first task. This confirms that the objective **O1** (forgetting) has also been attained.

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.



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

Figure 5: Test accuracy on clean testing images for **Split-CIFAR-10**

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 down to 0% after being poisoned. During the current task, however, the accuracy can still achieve the same level as in Figure 5a.

F.2 Investigation of the trigger-discarding property

To better illustrate property **P4**, we set up two experiments on Split-MNIST using cWGAN.

The **first** experiment 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 [49].

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.

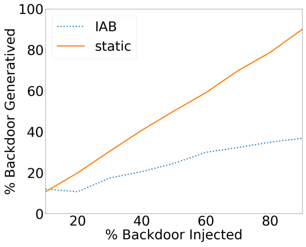
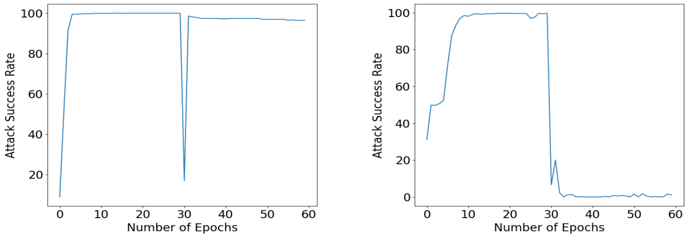


Figure 6: Replayed Backdoor



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

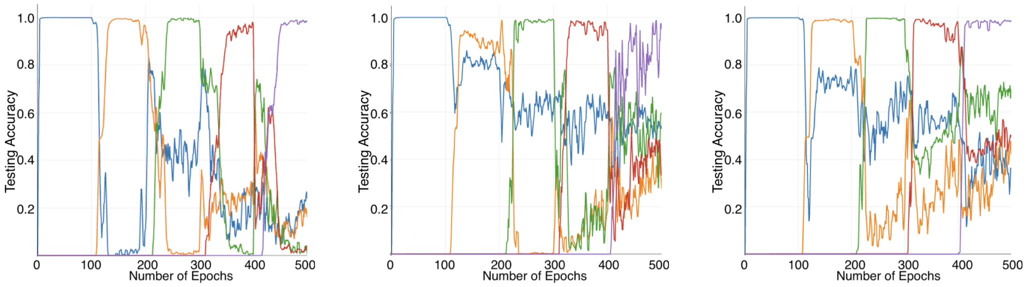
Figure 7: ASR for static backdoor and IAB

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.

F.3 Assessing CIAP attack under defense (objective O3)

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

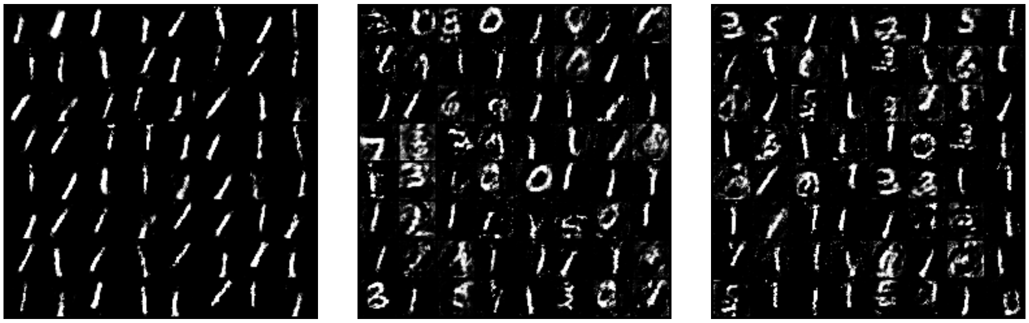


(a) $\nu = 0.15$ (b) $\nu = 0.25$ (c) $\nu = 0.35$

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

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%.

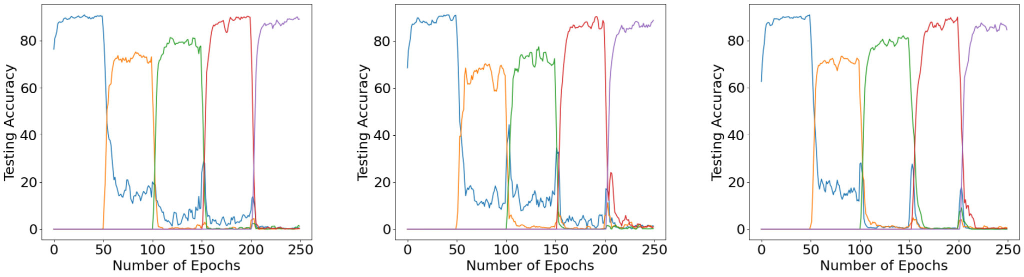
The limited improvement could be partially ascribed to the compromised image quality due to the backdoors. To better visualize the consequence of poisoning on the replayer, we compare in Figure 9 the replayed images before and after the attack. Figure 9a presents example images generated by a replayer 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.



(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 ($\nu = 0.25$).

Similar to the experiment on Split-MNIST, we tested the attack with ν-SVM defense on split-CIFAR-10. As shown in Figure 10, 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 10: Test accuracy after defense with different ν values for **Split-CIFAR-10**

G Conclusion and Future Work

We proposed a novel poisoning attack on the generative replayer in continual learning, so that catastrophic 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 can not 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, which is promising because the generators G_t can be aggregated into a single one.

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.