ContinuousPoisoning: Assessing the Claims and Security Aspects of Continual Learning

Written Report

Sneha Almeida

Matrikel Number: 23036892

Friedrich-Alexander-Universität
Technische Fakultät

1 Introduction

In this work, we have reproduced the results in the paper titled "Targeted Data Poisoning Attacks Against Continual Learning Neural Networks" by implementing it.

As neural networks are trained on newer and more recent data, they tend to abruptly forget previously learned knowledge, thus making the trained network as useless as (or more useless than) a random guess. This abrupt forgetting resulting from training on more and more data is called "catastrophic forgetting". The concept of continual learning helps overcome catastrophic forgetting. In continual learning, we constantly update our understanding of the world by continuously obtaining new data while making sure that the phenomenon where previously learned knowledge is forgotten is avoided. In this approach, we encounter a sequence of experiences over time. We do not have access to all the experiences simultaneously as we get data (in the form of experiences) in real time while training the network.

As continual learning emerges as a new paradigm, in this work, we test this paradigm's security by subjecting various forms of continual learning to data poisoning attacks and observing the change produced in performance as a result of the attacks. We perform a series of experiments as mentioned (and performed) in the paper under consideration [6], and try to evaluate how susceptible the continual learning paradigm is to data poisoning attacks. We evaluate the harm that data poisoning can do to this paradigm because, as mentioned earlier, data arrives in a streaming fashion. Hence, it can be easily tampered with while it is being streamed. We also compare our results with [6] to see if we get any deviations in the results that have been reported, and if there are any deviations then we try to find the causes for those deviations. Also, though we try to replicate as many aspects of [6] as possible, some aspects mentioned in the paper are too vague and we had to make assumptions during implementations. Nevertheless, all the assumptions have been clearly stated in 2.5

Implementation of this work is available at `https://github.com/sneha-almeida/Adversarial-Attack-Continual-learning/`

1.1 Related Terminologies

Here we will discuss various topics, terminologies, and their relation to our work. Let's have a look at the following terminologies:

Continual Learning

Continual learning can be summarized as follows: "Learning a mapping function and then updating it over time-based on the arrival of data over that period" [6]. In continual learning, data arrives in streams. Each of these streams is called an experience. There can be an infinite number of experiences. Let's denote these experiences with e_i where i denotes the number of experiences. There are 2 types of continual learning scenarios, namely, *Domain Incremental Continual Learning* and *Task Incremental Continual Learning*. These have been explained further in this section.

Scenario

A scenario is a specification of the continual stream of data [1]. In other words, it specifies what type of data will come in as a sequence of experiences. We are considering two scenarios, namely, the task incremental learning scenario and the domain incremental learning scenario.

Domain Incremental Continual Learning

A deep learning model is trained on multiple tasks in continual learning. In the Domain Incremental Continual Learning, the tasks are all similar (they have the same class labels). So, it is not necessary to identify the tasks individually or to distinguish them. Mathematically, this can be expressed as: $f : x \rightarrow y$, where $x \in X$: input, $y \in Y$: class label. Structurally, the problem is the same, but the context of input distribution changes. We are using the RotatedMNIST dataset [6] which is essentially just the MNIST dataset

rotated at different angles. We are considering the following angles here: $0^{\circ}, 40^{\circ}, 80^{\circ}, 120^{\circ}, 160^{\circ}$. Each angle of rotation is a different task. Sample images for each of these tasks from the dataset have been provided in figure 1. Hence, just the orientation of the dataset is changed in every task and hence, the class labels (which are the digits in the image) remain the same.

Task Incremental Continual Learning

As mentioned earlier, a model is trained on multiple tasks in continual learning. In task incremental continual learning, the tasks are identifiable to the concerned algorithm. This is because either the tasks are distinguishable or the tasks are provided with unique task IDs or labels. Mathematically, this can be expressed as: $f : x \times C \rightarrow y$, where $x \in X$: input, $y \in Y$: class label, $c \in c$: Task ID. Here the tasks are completely different from each other (different labels and sometimes even different numbers of classes). We are using the MNIST fellowship [6] dataset to perform this type of incremental learning. The MNIST fellowship dataset is a collection of 3 tasks, namely the original MNIST dataset for digit classification, FashionMNIST, which is the classification of fashion items, and KMNIST which is the classification of handwritten characters. Sample images for each of these tasks from the dataset have been provided in figure 2. All these tasks are completely different from one another and hence have to be identified differently (we are using unique numerical identifiers for each class).

Benchmark

A benchmark is a well-defined and carefully thought combination of a scenario with one or more datasets [1]. We have the following benchmarks in our experiments:

- Benchmark 1: Domain Incremental (scenario) and RotatedMNIST (dataset)
- Benchmark 2: Task Incremental (scenario) and MNIST Fellowship(dataset)

Adversarial Attack

An adversarial attack is an attack on a system that either renders the system completely unusable or compromises the performance of the system so that the trust in the system is reduced. There are 2 major types of adversarial attacks, namely evasion attacks and poisoning attacks [6]. We are performing poisoning attacks in our series of experiments. In this type of adversarial attack, the available data is tampered with so that the malicious objectives of the attacker can be met. In our case, the objective is to lower the accuracy of a target task without tampering with the data for that task.

Label Flipping

Label flipping is the most common poisoning attack on a classification-based Machine Learning model. As

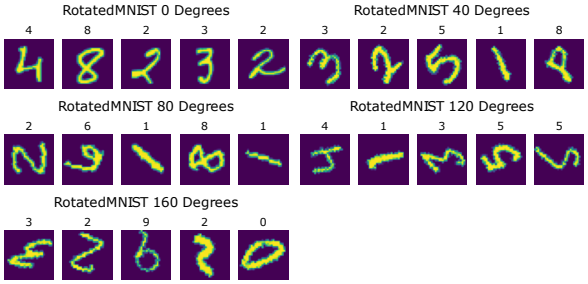

Fig. 1: Sample images from the RotatedMNIST dataset. There are 5 images for each rotation and the same MNIST Dataset is used by rotating it at specified angles.

the name suggests, we change the labels in the dataset randomly before training the model on that data. So, the model is now trained on wrong data-label combinations and it will have a very low performance. We use the following formula to flip the labels of 10% data in the dataset:

$poisoned\_label = original\_label + random(1, n)\%n$

where n is the total number of classes in the experience, random (x,y) is the function that selects a random integer between x and y.

This formula ensures that the correct label is never assigned to the corresponding data point.

2 Methodology

Three environments were created to compare results in each environment. The skeletal structure of these environments is common. There are **two scenarios** in each environment, namely **the Task Incremental learning scenario** and **the Domain Incremental learning scenario**. As mentioned in 1.1, the task incremental learning scenario is based on the RotatedMnist dataset [6] and the Domain incremental learning scenario is based on the MNIST fellowship [6] dataset. Each of these scenarios applies **three different strategies** to the concerned dataset, namely, **Synaptic Intelligence (SI)** [3] [7], **Elastic Weight Consolidation (EWC)** [7], and **Online Elastic Weight Consolidation (OnlineEWC)** [7]. First, the model is trained on each of these strategies, and then testing is performed on the trained models.

2.1 Data Arrival in Continual Learning

Data arrives in streams over time. These streams are called experiences, which can be theoretically infinite. The model is updated with every experience that arrives. All the experiences collectively form a sequence which can be represented by:

$S = e_1, e_2, ...e_n$, where e is the experience, n is the total number of experiences, and S is the sequence of all experiences. We are dealing with supervised classification problems in both scenarios

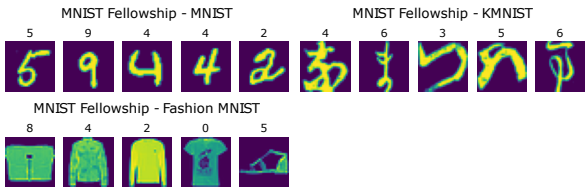

Fig. 2: Sample images from the MNIST Fellowship dataset. There are 5 images for each dataset, namely MNIST, KMNIST and FashionMNIST, which together form the MNIST Fellowship dataset.

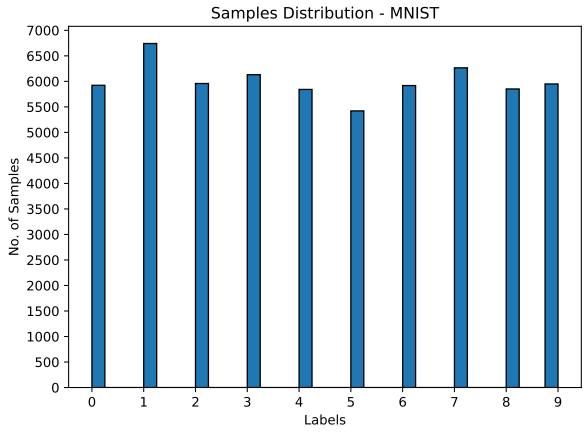

Fig. 3: Label Distribution MNIST dataset. KMNIST and FashionMNIST each consists of 10 classes with 6000 images per class

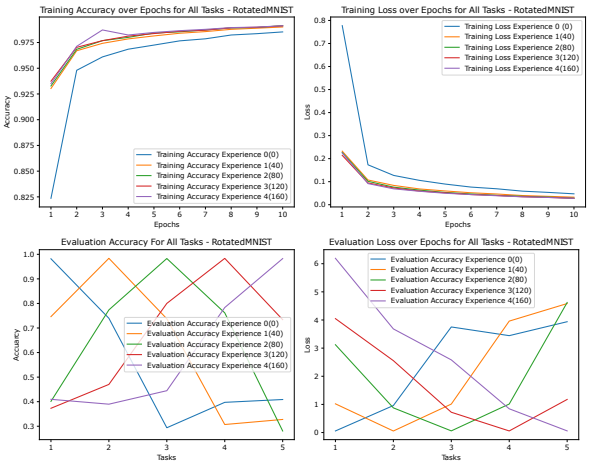

Fig. 4: Training and Evaluation Plots for Loss and Accuracy on RotatedMNIST. Top left: Training Accuracy over 10 Epochs for all 5 tasks, Top Right: Training Loss over 10 Epochs for all 5 tasks, bottom left: Evaluation Accuracy for all 5 tasks, bottom right: Evaluation Loss for all 5 tasks

(task incremental learning scenario as well as domain incremental learning scenario). For a supervised classification problem, each experience consists of a batch of samples where each sample is a tuple containing an input and a target label which can be represented as $< x_i^k, y_i^k >$, where i is the experience number and k represents k^{th} sample in the experience.

In our experiments, as mentioned earlier, we have two scenarios: domain incremental learning and task incremental learning. In domain incremental learning, we are using the RotatedMNIST dataset. This dataset consists of 5 experiences. Each experience consists of the MNIST dataset rotated by a certain angle (angles are mentioned in 1.1). In task incremental learning, we are using the MNIST Fellowship dataset. MNIST fellowship dataset is a collection of 3 datasets, namely, KMNIST, FashionMNIST, and Original MNIST datasets [7]. These 3 datasets form 3 experiences for the task incremental learning scenario.

2.2 Training and Model Evaluation in Continual Learning

As mentioned in section 2.1, the model is updated continuously as experiences arrive. It is a machine-learning model that is equipped with a continual learning strategy [6]. There are three strategies that we are using in each of the scenarios, namely Synaptic Intelligence (SI), Elastic Weight Consolidation (EWC), and Online Elastic Weight Consolidation (Online EWC). We first create a simple Multi-Layer Perceptron model and then apply one strategy at a time to that model. Results differ according to the type of strategy applied to the model (even if the base model and all its parameters remain the same).

Implementation Detail of the Machine Learning Model are specified below:

- Optimizer = Adam
- Learning Rate = 0.0001
- Batch Size = 128
- Eposch = 10
- Regularization factor for **Task Incremental Learning** :
  - SI = 5
  - EWC = 5000
  - Online EWC = 5000
- Regularization factor for **Domain Incremental Learning** :
  - SI = 5
  - EWC = 750
  - Online EWC = 1000
- Step Size for Adversarial Poisoning Algorithm

- SI = $\frac{4.\epsilon}{T}$, where ϵ is the size of perturbation and T is the number of iterations in the adversarial poisoning algorithm
- EWC = $\frac{2}{255}$
- Online EWC = $\frac{4.\epsilon}{T}$, where ϵ and T have the same relevance as above

Once the model has been trained according to a specific strategy, the model is evaluated on the test experiences (streams contain train experiences and test experiences), and the performance of the model is calculated based on the loss and accuracy of the trained model. The metrics that we consider are Loss and Accuracy associated with each epoch in the experience and those associated with the overall experience.

2.3 Environments

As mentioned earlier, We have created 3 environments for conducting experiments and comparing results within each environment and with other environments. Various models are trained in each environment and their performance is compared in section 3.

Defender Environment
This is the original continual learning environment. Its structure is similar to the skeletal environment described at the beginning of section 2. This is the basic environment. Other environments are an extension of this one. It demonstrates a simple continual learning paradigm with 2 scenarios: task incremental learning scenario and domain incremental learning scenario, where each scenario implements 3 strategies, SI, EWC, and Online EWC separately.

Attacker Environment: Label Flipping attack
This environment is an extension of the basic defender environment. Here we target one task in each scenario and the goal is to lower the accuracy of the target task in that scenario, without manipulating the inputs or labels in that task (to maintain stealthiness). In the first scenario with domain incremental learning on the RotatedMNIST dataset, we target the task learning the representation of the MNIST dataset with no rotation or rotation of 0°. In the next scenario, with task incremental learning on the MNIST Fellowship dataset, we target the task that learns the representation of the MNIST dataset. The process of performing the attack in both scenarios is the same. Label flipping attack has to occur before the experience streams are created. It is designed in such a way as to mimic a situation in which an attacker can flip the target labels while the streams of experiences are sent to the model for training. We performed the following steps for label flipping attack:

1. We picked 10 % of the samples from the first task (or experience, we will use the term task henceforth).
2. We then flipped the labels of those samples as explained in label flipping in section 1.1.
3. Then the data points (in our case, images) and the flipped labels were injected into other tasks. So other tasks had 10% more samples than the target task.
4. Further, experiences are created and then the models are trained on this poisoned data.

Initially, the target task will be trained. When the target task is trained, the model performs very well on that task. Later as other tasks are trained, the performance of the model on the target task decreases because other tasks have been injected with the wrong representations of the target task (along with the correct representations of other tasks) and the model has been forced to learn the wrong representations of the target task. This attack is, however, not very subtle, thus making it easy to detect, because the number of samples in the non-target tasks is more than what it has to be. Also, the labels have been flipped, which can also be detected by closer examination. Since this is a dirty label attack, it is not as stealthy as its clean label counterparts. More details about the implementation of this attack can be found in section 2.4.3.

Attacker Environment: Adversarial Poisoning Attack
This environment is an extension of the previous environments. In this environment, we perform the adversarial attack on the continual learning defender environment using some aspects of label flipping. However, unlike label flipping, this is a clean label attack which makes it difficult to detect the attack, thus making the attack stealthy. The goal of this attack is the same as that of label flipping: *"reduce the accuracy of the target task using data from other tasks"* (the data of the target task has not been tampered with in any way). The target tasks are the same for both scenarios as in the label-flipping environment. In this attack, based on the fraction of data perturbed, there are 4 cases, namely 5%, 10%, 15%, and 20% perturbation in the non-target tasks. Details of the implementation of this adversarial attack algorithm are given in 2.4.4.

2.4 Implementation details

We performed all of our experiments on Google Colab, which provides an online platform for executing Python code in the form of ipython (or .IPYNB) notebooks [8]. We have 3 IPYNB notebooks, one for each environment (refer section 2.3 for environments). We mostly used the following libraries:

- Avalanche
  End-to-end Python library for continual learning based on Pytorch [9]. This library can simulate continuous learning environments in a Python program or an IPYNB notebook.

- Pytorch
  The Avalanche library is not only based on the Pytorch library but also supports many Pytorch functionalities. Hence, we have also used the Pytorch library.

Implementation Details of major tasks have been highlighted further

2.4.1 Creation of Experience

Data arrival has been explained in section 2.1. Here we will explain the steps that we performed from an implementation perspective for the creation of an experience. To begin with, we have individual datasets i.e., for domain incremental learning, we have the MNIST dataset, which we have to rotate at pre-specified angles to obtain the RotatedMNIST dataset and we have FashionMNIST, KMNIST, and MNIST datasets separately which we have to combine in an appropriate way to obtain the MNIST Fellowship dataset. The steps involved in creating an experience are as follows:

1. Transform the data available in respective datasets(RotatedMNIST or MNIST Fellowship). These transformations can include conversion to tensors, conversion to PIL images, and rotations. Here, training and testing data were kept separate.

2. Convert each task into an "AvalancheDataset" object [9] and give task labels to it. Here we are referring to individual tasks like FashionMNIST, MNIST, KMNIST, or the MNIST dataset rotated at a specific angle. For the Domain incremental scenario, we should have 5 such datasets (one dataset for each degree of rotation). For the Task incremental scenario, we should have 3 datasets, namely MNIST, KMNIST, and FashionMNIST.

3. Create a dataset benchmark for each scenario, namely task incremental learning and domain incremental learning. We create 2 benchmarks as mentioned in 1.1.

4. Benchmark contains train and test streams. These streams have to be recovered from the benchmark.

5. Streams contain experiences. Experiences can be obtained by iterating over streams. These experiences are used for training the models.

2.4.2 Training and Evaluation of Continual Learning Models

The architectural details of the neural networks are as specified in [6]. The target tasks have been trained first in both scenarios and all the 3 environments, followed by non-target tasks. The order of training all the tasks across both scenarios is the same across all environments.
Accuracy and loss are the metrics we use to measure performance. We save these metrics (from every task) for each scenario in all environments to CSV files to compare them. These have been tabulated in the tables 2, 3, 4, 5, 6.

2.4.3 Label Flipping Attack

While we have a theoretical overview of the process of label flipping from section 2.3, here we present a more implementation-oriented summary of how the label flipping attack is performed using Avalanche and Pytorch in Python. The steps are as follows:

1. We first obtain the required datasets, i.e., MNIST, KMNIST, and FashionMNIST. Then we create the datasets that we need for our experiments namely, rotated MNIST, by rotating the MNIST dataset at 5 different angles as specified in section 1.1 and MNIST fellowship by combining MNIST, KMNIST, and FashionMNIST.

2. We select a target task from each scenario. In the Domain Incremental Learning scenario, which is based on the RotatedMNIST dataset, we target the task with 0° rotation. Similarly, in the Task Incremental Learning scenario that is based on the MNIST Fellowship dataset, we target the MNIST classification task.
   Here, we are selecting target tasks because we will lower the performance of the target task by flipping labels of data in the target task and injecting it in the non-target task. Also, we train the model on the target task before training it on other (poisoned) tasks.

3. We select samples equivalent to 10% from the target task. (This selection consists of both inputs and labels).

4. The labels of these selected samples are then flipped according to the formula given in section 1.1. Then these samples (data points and labels) are converted into a list.

5. This list of samples from the target task is then converted to an "*Avalanche dataset*" object [9].

6. We have chosen 10% samples from the target task to inject those into other tasks (in both scenarios). So, the other tasks will have more samples than the target task. A sample in the dataset is represented as: (input, input label, task label). We have the input and the input label converted into an "*Avalanche dataset*" object. Now we create task labels which will be used for representing our samples from the target task into other tasks when they are injected into other tasks.

7. Now, we finally inject the additional samples taken from the target task into other tasks by

creating a classification dataset consisting of the original data in that task, Avalanche dataset of flipped samples created in step 5, and task labels created in step 6.

Now, all the tasks in the dataset, except for the target task have been injected with 10% samples from the target task whose labels have been flipped. So, while the model is being trained to learn representations of the non-target task, it is also learning the wrong representation of the target task, on which it has already been trained. In this way, the model is being poisoned during training.

2.4.4 Adversarial Attack

The adversarial attack is performed before the models can be trained. As mentioned earlier, this attack changes the data points in the dataset to have the same effect as label flipping. But this is a clean label attack as we are neither adding more data points nor changing the labels of the data points. This makes the attack more stealthy. Here is a step-wise summary of the adversarial attack algorithm: First, let ϵ represent the fraction of the dataset to be perturbed. Further, we call the non-target dataset to be perturbed as the poison dataset.

1. Train clean models on all three strategies, namely SI, EWC, and Online EWC on the datasets available before the poisoning.
2. The product of the length of the poison dataset and ϵ defines the number of data points that are randomly selected as poison samples.
3. Calculate the gradients for these points based on the clean models being trained. These gradients are called *target gradients*.
4. Calculate gradients for all the data points in the poison dataset. These gradients are called *poison gradients*.
5. Calculate cosine similarity from target and poison gradients
6. Calculate momentum based on the cosine similarity
7. Update all images in the poison dataset based on the momentum
8. Repeat steps 4 to 7 for 240 iterations.
9. Return the poisoned dataset

All the details of mathematical implementations in the algorithm (like formulae) have been taken from the original work in [6].

2.5 Assumptions

This was an attempt to replicate the work titled "Targeted Data Poisoning Attacks Against Continual Learning Neural Networks". As the implementation was not available publicly, we had to make some assumptions while replicating this work. Those assumptions have been mentioned below:

- The Process of Computation of gradients required in the adversarial attack algorithm is not mentioned explicitly in the work under consideration. We assumed that the gradients are calculated using a model trained on the clean dataset. Hence, we also assume that the attacker can access the clean data and the model parameters to train a clean model on the data.
- In label flipping, data poisoning occurs before data streams are created. We had to assume this because it was not mentioned when data is exactly poisoned for label flipping.

3 Results And Discussion

The experiments performed have been summarized in table 1

After performing experiments mentioned in the paper in the defender environment for domain incremental and task incremental scenarios, we realized that the parameters specified for the respective experiments did not give the results that matched those obtained in [6]. These results are tabulated in tables 2 and 3. The model exhibited catastrophic forgetting to an extent of concern and the performance of the model on initial tasks was heavily compromised. In other words, the model forgets what it has learned in the previous tasks after learning new tasks. We suspected the reason to be the regularization parameter (responsible for a stability-plasticity trade-off of the model) is not being tuned correctly. So, we performed some more experiments where we tried to change the value of this parameter for the SI Strategy in the domain incremental learning scenario. The results are tabulated in table 4. These results indicate that changing the value of the regularization parameter does not affect the model's performance.
Nevertheless, we performed label flipping by changing the labels of 10% data samples according to the procedure mentioned in 1.1 for domain incremental and task incremental scenarios. The results of label flipping are tabulated in tables 2 and 3.
Further, as we experimented in the Adversarial Attack Environment, we realized that the computation capabilities required to poison the existing amount of data are very high. So we reduced the number of samples by a huge margin. We performed the experiments on 1200 samples for the domain incremental scenario. The results are tabulated in table

	Domain Incremental	Task Incremental
Defender Environment	• Experiments mentioned in the paper • Experiments with regularization parameters for SI Strategy	Experiments mentioned in the paper
Label Flipping Environment	Flip labels of 10% data as mentioned in the paper	
Adversarial Attack Environment	• 1200 samples for 5%, 10%, 15%, 20% perturbations • 2400 samples for 5%, 10%, 15%, 20% perturbations	–

Tab. 1: Experiments Performed

5. Since the results were unsatisfactory, we increased the number of samples to 2400 in each dataset and repeated the experiments as mentioned earlier in hopes of better results, but the results did not improve considerably. These results are mentioned in table 6. While we tried to vary the number of samples to see if there were any considerable changes in the results, [6] does not contain any information on the number of samples used to train the models in any environment.

So, instead of experimenting with the poisoning algorithm on both task incremental and domain incremental scenarios for all the available samples in the datasets, we performed our experiments on the domain incremental scenario with varying numbers of samples in each set of experiments. Due to time constraints, we could not perform these experiments on the task incremental scenario.

3.1 Deviation from Results

Heavy deviations were observed in the results obtained from our experiments against the results reported in [6]. We suspect the following reasons for the deviations observed:

- Implementation of the work in [6] is not publicly available. There might be differences in the implementations that led to the deviations.
- It was observed that catastrophic forgetting was prominent in clean models (which were trained on clean data in the defender environment). This catastrophic forgetting also made its way into the poisoning algorithm because the poisoning algorithm uses clean models for the calculation of gradients.
- Parameters other than plasticity-stability trade-off might need tuning.

4 Conclusion

In this work, we attempted to replicate the results of [6]. However, with our setup, we could not obtain results that matched. However, we performed experiments in various directions, like increasing the number of data points in the dataset and changing the regularization parameter of the model to adjust the stability-plasticity trade-off, to find the root cause of deviations from the original results so that we could fix it, but we could not pinpoint to a specific factor as a root cause, based on our investigations.

5 Future Work

We propose some extensions to the work that has been performed.
Currently, each experience comprises data from one task. Because of this, when the model is trained on more recent experiences, it tends to forget previous

EWC		
Task	Clean	10% Label Flipping
1	53.17	1.69%
2	29.62	34.52%
3	87.61	49.31%
Online EWC		
Task	Clean	10% Label Flipping
1	50.60%	2.47%
2	37.13%	44.46%
3	89.18%	82.99%
SI		
Task	Clean	10% Label Flipping
1	20.46%	1.17%
2	29.31%	36.25%
3	88.13%	87.87%

Tab. 2: Evaluation Accuracy on MNIST Fellowship dataset for Clean Data and Data with Flipped Labels
Note: Task 1 was the targeted task during label flipping

EWC		
Task	Clean	10% Label Flipping
1	39.47%	17.64%
2	55.05%	71.74%
3	61.58%	70.79%
4	75.79%	78.26%
5	94.86%	94.32%
Online EWC		
Task	Clean	10% Label Flipping
1	43.26%	19.85%
2	59.44%	71.27%
3	61.68%	69.91%
4	74.70%	76.73%
5	94.90%	94.04%
SI		
Task	Clean	10% Label Flipping
1	33.37%	3.31%
2	32.64%	37.73%
3	38.24%	42.86%
4	72.38%	78.85%
5	98.04%	97.89%

Tab. 3: Evaluation Accuracy on RotatedMNIST dataset for Clean Data and Data with Flipped Labels
Note: Task 1 was the targeted task during label flipping

experiences. Since each experience is a different task, as the model learns new tasks, it tends to forget the old ones and this is termed "catastrophic forgetting" [6]. The first extension that we propose to the existing work is that the solution to the problem of catastrophic forgetting would be to distribute data from multiple tasks over one experience [10]. With such implementation, even if the problem of catastrophic forgetting exists, the model would never forget a complete task. It would forget some instances of all the tasks that it is being trained upon. While this approach helps overcome the issue with continual learning strategies where they forget old tasks, we have also identified some limitations to it. Those limitations are as follows:

- New tasks cannot not be added later:
  All the tasks on which the network is to be trained need to be known before the training starts. If new tasks are added in some later experience, then catastrophic forgetting of the old tasks is highly probable.
- Reduced generalization capability:
  Since the continual learning approach still forgets previously learned representations, it is not likely to forget an entire task, but it will forget the representations over all the tasks that it learned in the very earlier experiences. This will reduce the approach's generalization capability over all the tasks under consideration.

Another extension that we can have to this work can be described as follows: The adversarial attack algorithm perturbs the dataset in a specific way. This perturbed dataset is used to train a model. Since the perturbed dataset is used for training a model, it is a poisoning attack. Another extension to the work can be that if we use this perturbed dataset while testing, then we can perform an evasion attack. Now, let's assume that we have a trained model (which has been trained on a non-perturbed dataset). Another assumption is that the dataset perturbations are not visible to the human eye. Then, we can use the perturbed dataset samples for testing to forcefully demonstrate that the model performance is low on the testing dataset i.e., the model does not perform well on external datasets and hence, has poor generalization capability. This idea just looks at using the perturbed dataset for another malicious task.

While the two approaches presented above are extensions to the existing work, it is also worthwhile to look at why agreeable results could not be reproduced. Evaluating the current setup could be a starting point for further work.

References

[1] Lomonaco, Vincenzo, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, et al. "Avalanche: An End-to-End Library for Continual Learning." In 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 3595–3605. Nashville, TN, USA: IEEE, 2021. https://doi.org/10.1109/CVPRW53098.2021.00399.

[2] Dong, Yinpeng, Fangzhou Liao, Tianyu Pang,

SI					
Task	Task 1	Task 2	Task 3	Task 4	Task 5
λ = 10	34.17%	27.57%	30.13%	71.88%	97.79%
λ = 50	34.36%	26.84%	32.08%	71.29%	97.82%
λ = 500	33.56%	28.54%	32.24%	73.48%	97.82%
λ = 5000	34.41%	28.24%	32.68%	71.10%	97.83%
λ = 50000	34.91%	25.29%	34.21%	70.86%	97.98%
λ = 1	33.09%	27.83%	33.05%	71.26%	97.87%
λ = 0.1	34.84%	27.03%	33.01%	75.71%	97.88%
λ =0.01	32.04%	25.38%	32.03%	74.62%	97.72%
λ = 0.001	34.66%	24.36%	31.95%	74.98%	97.68%
λ = 0.0001	34.91%	26.87%	31.20%	75.02%	97.79%

Tab. 4: Evaluation Accuracy on RotatedMNIST Dataset for varying values of the regularization parameter for SI Strategy

EWC					
Task	Clean	5% poisoned	10% poisoned	15% poisoned	20% poisoned
1	29%	29%	22.5%	31.5%	30%
2	41%	32.5%	34.5%	42%	43.5%
3	58%	53%	50.5%	58.5%	57.5%
4	71.5%	75%	72%	74.5%	73.5%
5	84%	77.5%	81.5%	80%	77.5%
Online EWC					
Task	Clean	5% poisoned	10% poisoned	15% poisoned	20% poisoned
1	27%	28.5%	25%	31.5%	28.5%
2	39%	35.5%	34.5%	43%	42.5%
3	57.5%	54%	46.5%	60%	58%
4	74%	72.5%	68.5%	76%	74.5%
5	81.5%	78.5%	82.5%	79%	80.5%
SI					
Task	Clean	5% poisoned	10% poisoned	15% poisoned	20% poisoned
1	27%	19.5%	18.5%	24.5%	23%
2	36%	28%	28%	33.5%	31%
3	54%	50%	48%	51.5%	55%
4	72%	76.5%	70%	74.5%	73.5%
5	82.5%	79%	82.5%	82%	81.5%

Tab. 5: Evaluation Accuracy on Rotated MNIST for 1200 samples
Note: Task 1 was the targeted task during poisoning

EWC					
Task	Clean	5% poisoned	10% poisoned	15% poisoned	20% poisoned
1	25.25%	28.25%	27.5%	27.25%	29%
2	31.75%	26.25%	25.25%	34.75%	33%
3	41.5%	45.25%	38.75%	48.75%	46.5%
4	70.25%	76.25%	67.75%	70.25%	70.5%
5	86.25%	89.75%	84%	83%	85.25%
Online EWC					
Task	Clean	5% poisoned	10% poisoned	15% poisoned	20% poisoned
1	26.75%	29%	27.25%	26.25%	30.75%
2	31.75%	28%	25.75%	35.5%	33.25%
3	42.5%	45.25%	39.75%	48%	44%
4	73.5%	74%	65.75%	69.75%	71.25%
5	88%	90.5%	85%	84.75%	84%
SI					
Task	Clean	5% poisoned	10% poisoned	15% poisoned	20% poisoned
1	19%	21.5%	20.5%	23%	20.75%
2	20.5%	15.75%	16.75%	22.5%	20.75%
3	37.75%	38%	34.25%	41%	38.25%
4	72.25%	75.25%	69.5%	70.75%	72.5%
5	88.75%	90.25%	84.75%	85.75%	85.75

Tab. 6: Evaluation Accuracy on Rotated MNIST for 2400 samples
Note: Task 1 was the targeted task during poisoning

Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. “Boosting Adversarial Attacks with Momentum.” In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 9185–93. Salt Lake City, UT: IEEE, 2018. https://doi.org/10.1109/CVPR.2018.00957 .

[3] Zenke, Friedemann, Ben Poole, and Surya Ganguli. “Continual Learning Through Synaptic Intelligence,” 2017. https://doi.org/10.48550/ARXIV.1703.04200.

[4] Maltoni, Davide, and Vincenzo Lomonaco. “Continuous Learning in Single-Incremental-Task Scenarios,” 2018. https://doi.org/10.48550/ARXIV.1806.08568.

[5] Khan, Hikmat, Pir Masoom Shah, Syed Farhan Alam Zaidi, and Saif ul Islam. “Susceptibility of Continual Learning Against Adversarial Attacks,” 2022. https://doi.org/10.48550/ARXIV.2207.05225.

[6] Li, Huayu, and Gregory Ditzler. “Targeted Data Poisoning Attacks Against Continual Learning Neural Networks.” In 2022 International Joint Conference on Neural Networks (IJCNN), 1–8. Padua, Italy: IEEE, 2022. https://doi.org/10.1109/IJCNN55064.2022.9892774.

[7] Van De Ven, Gido M., Tinne Tuytelaars, and Andreas S. Tolias. “Three Types of Incremental Learning.” Nature Machine Intelligence 4, no. 12 (December 5, 2022): 1185–97. https://doi.org/10.1038/s42256-022-00568-3 .

[8] Bisong, E. and Bisong, E., 2019. Google colaboratory. Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners, pp.59-64.

[9] Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T.L., De Lange, M., Masana, M., Pomponi, J., Van de Ven, G.M. and Mundt, M., 2021. Avalanche: an end-to-end library for continual learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3600-3610).

[10] Hanul Shin, Jung Kwon Lee, Jaehong Kim, Jiwon Kim. “Continual Learning with Deep Generative Replay”. In *Advances in Neural Information Processing Systems*, vol. 30, 2017.