

Rowan University

Rowan Digital Works

Theses and Dissertations

6-6-2023

ADVERSARY AWARE CONTINUAL LEARNING

Muhammad Umer
Rowan University

Follow this and additional works at: <https://rdw.rowan.edu/etd>



Part of the [Artificial Intelligence and Robotics Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Umer, Muhammad, "ADVERSARY AWARE CONTINUAL LEARNING" (2023). *Theses and Dissertations*. 3130.
<https://rdw.rowan.edu/etd/3130>

This Dissertation is brought to you for free and open access by Rowan Digital Works. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Rowan Digital Works. For more information, please contact graduateresearch@rowan.edu.

ADVERSARY AWARE CONTINUAL LEARNING

by
Muhammad Umer

A Dissertation

Submitted to the
Department of Electrical & Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Doctor of Philosophy
at
Rowan University
April 26, 2023

Dissertation Chair: Robi Polikar, Ph.D., Professor & Department Head, Electrical & Computer Engineering Department, Rowan University

Committee Members:

Ravi Prakash Ramachandaran, Ph.D., Professor, Department of Electrical & Computer Engineering, Rowan University

Ghulam Rasool, Ph.D., Assistant Professor, Department of Electrical Engineering, University of South Florida

Shreekanth Mandayam, Ph.D., Professor Emeritus, Electrical & Computer Engineering Department, Rowan University

Umashanagar Thayasivam, Ph.D., Professor & Chair of Department of Science & Mathematics, Rowan University

Gregory Ditzler, Ph.D., Associate Professor, Electrical & Computer Engineering Department, Rowan University

Dedications

I would like to dedicate this work to my parents, and my family

Acknowledgements

I would like to thank my advisor and mentor, Professor Robi Polikar, whose zeal and vision in research have profoundly inspired me, and whose continual guidance, encouragement, and support have made this work possible. I am especially grateful to him for his trust and the freedom he gave me to pursue research on a topic of interest to me.

I also would like to extend my gratitude to Dr. Ravi Prakash Ramachandran, Dr. Umashanger Thayasivam, Dr. Shreekanth Mandayam, Dr. Ghulam Rasool, and Dr. Gregory Ditzler for being part of my committee and for their generous time, commitment, fruitful discussions, and insights.

Finally, I would like to thank my family who has always supported and encouraged me to continue moving forward throughout my academic career.

Abstract

Muhammad Umer
ADVERSARY AWARE CONTINUAL LEARNING
2022-2023

Robi Polikar, Ph.D., Professor & Department Head, Electrical & Computer Engineering
Department, Rowan University
Doctor of Philosophy

Continual learning approaches are useful as they help the model to learn new information (classes) sequentially, while also retaining the previously acquired information (classes). However, these approaches are adversary agnostic, i.e., they do not consider the possibility of malicious attacks. In this dissertation, we have demonstrated that continual learning approaches are extremely vulnerable to the adversarial backdoor attacks, where an intelligent adversary can introduce small amount of misinformation to the model in the form of imperceptible backdoor pattern during training to cause deliberate forgetting of a specific class at test time. We then propose a novel defensive framework to counter such an insidious attack where, we use the attacker’s primary strength – hiding the backdoor pattern by making it imperceptible to humans – against it and propose to learn a perceptible (stronger) pattern (also during the training) that can overpower the attacker’s imperceptible (weaker) pattern. We demonstrate the effectiveness of the proposed defensive mechanism through various commonly used replay-based (both generative and exact replay-based) continual learning algorithms using CIFAR-10, CIFAR-100, and MNIST benchmark datasets. Most noteworthy, we show that our proposed defensive framework considerably improves the robustness of continual learning algorithms with ZERO knowledge of the attacker’s target task, attacker’s target class, shape, size, and location of the attacker’s pattern. The proposed defensive framework also does not depend on the underlying continual learning algorithm. We term our proposed defensive framework as Adversary Aware Continual Learning (AACL).

Table of Contents

Abstract	v
List of Figures	ix
List of Tables	xi
Chapter 1: Introduction	1
1.1 Motivation: Robustness in Continual Learning	1
1.2 Research Contributions	3
1.3 Organization of The Dissertation	4
Chapter 2: Background	6
2.1 Continual Learning Framework	6
2.1.1 Continual Learning Scenarios	7
2.1.2 Relationship of Continual Learning with Domain Adaptation / Transfer Learning	10
2.2 Continual Learning Approaches	10
2.2.1 Architecture-Based Approaches	11
2.2.2 Regularization-Based Approaches	12
2.2.3 Replay-Based Approaches	13
2.3 Adversarial Machine Learning & Continual Learning	14
2.3.1 Backdoor Poisoning Attack	15
2.3.2 Conventional Backdoor Defenses & Their Limitations in Contin- ual Learning Setting	19

Table of Contents (Continued)

Chapter 3: Related Work: Poisoning Importance Weighting Based Domain Adaptation	22
3.1 Logistic Regression Based Importance Estimation.....	24
3.1.1 Poisoning Logistic Regression Based Importance Estimation	26
3.2 Unconstrained Least Squares Importance Fitting (uLSIF)	31
3.2.1 Poisoning Unconstrained Least Squares Importance Fitting	32
Chapter 4: False Memory Formation in Continual Learning (Exploring Vulnerabilities of Continual Learning Approaches)	37
4.1 False Memory Formation	37
4.1.1 Attacking Regularization-Based Continual Learning (CL) Approaches.....	40
4.1.2 Attacking Replay-Based Continual Learning (CL) Approaches	42
Chapter 5: Adversary Aware Continual Learning (AACL) Framework	48
Chapter 6: Experiments & Results	59
6.1 Attacking Importance Weighting Based Domain Adaptation	59
6.1.1 Attacking Logistic Regression Based Importance Estimation	59
6.1.2 Attacking Unconstrained Least Squares Based Importance Estimation (uLSIF)	70
6.2 Adversary Aware Continual Learning (AACL): Attacking and Defending Modern Continual Learning Approaches	79
6.2.1 Defending Exact Replay-Based Continual Learning Approaches using AACL	80

Table of Contents (Continued)

6.2.2 Defending Generative Replay-Based Continual Learning Approaches using AACL	91
Chapter 7: Conclusion & Future Work	95
7.1 Summary of Future Work.....	97
References	98
Appendix A: Derivations of Poisoning Importance Estimation.....	108
Appendix B: Attacking Regularization Based Continual Learning (CL) Approaches	115

List of Figures

Figure		Page
Figure 1.	Training With Backdoor Pattern.....	18
Figure 2.	Testing With Backdoor Pattern	18
Figure 3.	Backdoor Attack to Replay-Based CL Model	47
Figure 4.	Adversary Aware Continual Learning Defense for Replay-Based CL Model	55
Figure 5.	Training and Test Data Distributions.....	60
Figure 6.	Adapted Training Distribution Using LR Classifier Based Importance Estimation.....	61
Figure 7.	Poisoning Importance Estimation With Perfect Knowledge Attack ...	63
Figure 8.	Poisoning Importance Estimation With Limited Knowledge Attack ..	65
Figure 9.	2-Dimensional Training and Test Data Distributions for Poisoning Logistic Regression Based Importance Estimation.....	69
Figure 10.	Contour Plots Showing Training and Test Data Distributions for Poisoning Logistic Regression Based Importance Estimation.....	70
Figure 11.	Training and Test Data Distributions in First Scenario Figure 11a; Second Scenario Figure 11b; Third Scenario Figure 11c;	71
Figure 12.	Attacker's Objective for Scenario 3 (With Single Attack Point)	74
Figure 13.	Predicted Importance Values for Keystrokes Dataset	75
Figure 14.	Training (Black Contour) and Test (Red Contour) Data Marginal Distributions.....	76
Figure 15.	Traffic Dataset Samples.....	79
Figure 16.	Imperceptible Attack Pattern & Perceptible Defense Pattern for CIFAR-10 Images	81

List of Figures (Continued)

Figure	Page
Figure 17. Sample CIFAR-10 Images	82
Figure 18. Imperceptible Attack Pattern & Perceptible Defense Pattern for MNIST Images	87

List of Tables

Table		Page
Table 1.	Prediction Accuracy for Perfect Knowledge Attack Scenario	67
Table 2.	Prediction Accuracy for Limited Knowledge Attack Scenario	67
Table 3.	Mean Square Error (MSE) Between the True Importance Values and the Predicted Importance Values (No Attack)	72
Table 4.	Mean Square Error (MSE) Between the True Importance Values and the Predicted Importance Values (with Attack)	73
Table 5.	Prediction Accuracy in Percentage	77
Table 6.	Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-10	84
Table 7.	Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-100	86
Table 8.	Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on MNIST	88
Table 9.	Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-10 for Defending Task 2 to Task 5	89
Table 10.	Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-100 for Defending Task 2 to Task 10	90
Table 11.	Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on MNIST for Defending Task 2 to Task 5	91
Table 12.	Test Accuracy (in %) of Generative Replay-Based Continual Learning Approaches on MNIST	93
Table 13.	Test Accuracy (in %) of Generative Replay-Based Continual Learning Approaches on MNIST for Defending Task 2 to Task 5	94

Chapter 1

Introduction

1.1 Motivation: Robustness in Continual Learning

Continual learning (CL)—also referred to as life-long, sequential, or incremental learning—is the problem of learning tasks sequentially from an infinite stream of data [1]. In streaming data, different tasks arrive at different points in time and each task has a different distribution. The goal of the continual learning algorithm is to effectively adapt to the new task(s) under an evolving environment without forgetting the knowledge acquired in the previous task(s). Due to the sequential nature of the data, only the data from a single task or sometimes a few tasks are available to the CL model at once. In this context, a single task refers to a particular period of time during which the data come from a stationary distribution. A common phenomenon plaguing CL is that of *catastrophic forgetting* [2], where the level of performance acquired by the model on some previously-learned task is partially or completely lost (forgotten) while training to acquire new knowledge for learning a subsequent task. Catastrophic forgetting is often characterized by the *stability - plasticity dilemma* [3], where stability refers to the preservation of past knowledge, and plasticity refers to the ability to acquire and integrate new knowledge.

Many real-world problems are characterized by non-stationary data. Network intrusion, web usage and user interest analysis, natural language processing, speech and speaker identification, spam detection, anomaly detection, analysis of financial, climate, medical, energy demand, or pricing data, as well as the analysis of signals from autonomous robots and devices, brain signal analysis, and bio-informatics are just a few examples of the real world problems where underlying distributions may – and typically do – change over time. It is, therefore, of paramount importance to build machine learning models that can exhibit

continual learning in changing environments without forgetting most of the information already acquired.

Humans learn concepts sequentially, and do not suffer from the problem of catastrophic forgetting [4]. However, for common ML models, such as artificial neural networks (ANNs), sequential learning is challenging due to the potentially complete loss of previously acquired knowledge. It has been shown that the reason behind catastrophic forgetting in neural networks is the drastic change in their parameter values (that were optimally found for a specific task) when asked to learn a new different task [5]. One trivial solution to avoid catastrophic forgetting is to store all incoming examples and train the model from scratch, however such an approach is often computationally infeasible and impractical [6]. Ideally, data from all the previous tasks need not be stored in their raw format, and training from scratch should not be necessary for every incoming task in order to get an efficient and realistic artificial continual learning agent [7]. In fact, true continual learning requires the model to be able to acquire new knowledge only with access to the existing knowledge (current hypothesis) and the new data – but *not* the old data. As such, combining old and new data actually violates the true definition and intention of CL [1], [8], [9], [10].

As mentioned previously, the main challenge for continual learning is to avoid or lessen the impact of catastrophic forgetting. Therefore, much of the work in continual learning has focused on avoiding catastrophic forgetting while maintaining this delicate balance between stability and plasticity. In this work, we point out another and arguably more important challenge for continual / incremental learning algorithms, which is the vulnerability of these approaches to intentional adversarial attacks. While several CL approaches have been proposed over the years, these approaches have all been adversary agnostic, i.e., they have not considered the possibility of a malicious or adversarial attack. Yet, adversarial attacks on machine learning algorithms have been rapidly growing. Since continual learning is critical to learning from large, streaming datasets, the importance of studying the vulnerability of CL to adversarial attacks, as well as developing appropriate

defenses can be hardly overstated. In other words, neglecting the robustness aspect of continual learning approaches is imprudent. In this dissertation, I seek to answer the following two important questions: 1) can deliberate forgetting – or misinformation – be introduced in continual / incremental learning models through adversarial attacks; and 2) if so, how can continual / incremental learning models be protected and rendered more robust against such adversarial attacks?

1.2 Research Contributions

The primary focus of this effort is to propose a mechanism to ensure robustness in continual learning models against adversarial attacks. The core contributions and findings of this work are as follows

1. Before exploring the vulnerabilities of the modern continual / incremental learning approaches, we first proposed bi-level optimization-based poisoning attack strategy against related conventional co-variate shift-based domain adaptation approaches. We show that such approaches are extremely vulnerable to the proposed poisoning attack strategy. The attacker can assume complete control of these approaches by inserting a few strategically generated malicious samples into the training data.
2. We identified the vulnerabilities of continual learning approaches against poisoning attacks. Inspired by the misinformation effect in the human brains, we proposed an adversarial backdoor attack strategy to cause deliberate forgetting of a particular task or class of the attacker’s choice at test time. Most noteworthy, we show that the attacker can achieve its goal with great success even if the attacker’s backdoor pattern is imperceptible to human eye, and even if the backdoor pattern is provided to as few as just 1% of total training data of a single task.
3. In an adversarial setting, the attacker has a natural advantage, in part due to possible surprise element of the attack. The defender’s job is a bit more difficult. Hence,

one of the most important contributions of my dissertation involves exploring and proposing defensive mechanisms to make CL algorithms more robust against misinformation provided via backdoor malicious samples. Specifically, we propose a novel defensive framework to ensure robustness in continual learning models to imperceptible misinformation inserted via adversarial backdoor attack. Our defensive framework utilizes a small amount of defensive (decoy) samples to inhibit the impact of the malicious samples. The defensive (decoy) samples also contain a pattern similar to adversarial backdoor malicious samples but this pattern is: i) perceptible (stronger); and ii) different than the attacker’s unknown imperceptible (weak) pattern. The defensive samples are provided during the training of the CL model and thus serve to inoculate the CL model against the malicious samples. We referred to such learning or our defensive framework as the *Adversary Aware Continual Learning (AACL)*.

1.3 Organization of The Dissertation

Chapter 2 provides an overview and background for continual learning (CL), i.e., the framework of CL, different scenarios of CL, the relationship of CL with domain adaptation, and existing CL approaches. Chapter 2 also provides a brief overview of adversarial machine learning, false memory formation, backdoor poisoning attacks, the current defenses to backdoor attacks and their limitations in continual learning settings. Chapter 3 introduces the proposed poisoning attack strategy developed as part of this dissertation, to explore the vulnerabilities of domain adaptation approaches. Chapter 4 discusses the idea of extending adversarial backdoor attacks to the CL models to deliberately introduce the forgetting via imperceptible backdoor pattern. Chapter 5 introduces our proposed adversary aware continual learning framework to ensure robustness in the CL models. This chapter also discusses the similarities and differences of our proposed defense against one of the popular and well-known adversarial training defense. Chapter 6 presents the exper-

imental setup and results, demonstrating the vulnerabilities of domain adaptation and CL approaches to adversarial attacks. The promising results of our proposed defense framework to achieve robustness in CL models are also discussed in this chapter using various continual learning datasets and approaches. Finally, the conclusions and suggestions for future work are discussed in Chapter 7.

Chapter 2

Background

2.1 Continual Learning Framework

The continual learning (CL) setting that is normally considered in the literature learns from the different sequential tasks, where each task represent a separate supervised learning problem. The data associated with a single task is received at a time to a CL model. We also consider the same CL setting in this work. Each task presented to the CL model has different distributions however, the i.i.d. assumption is maintained *within* each task. The training for each task can thus be performed offline where shuffling and repeated revisiting of i.i.d. training data can be done to achieve reasonable training performance for a given task. When a task t is received with its training data and corresponding labels, (X^t, y^t) drawn from a distribution $P_t(X^t, y^t)$, the goal of continual learning is to find the optimal parameters θ^* that minimize the empirical risk of the overall model not only on the current task but also on all tasks $k = 1, \dots, t$ seen so far including the current task t . The empirical risk for this setting can then be written as [1]:

$$\sum_{k=1}^t \mathbb{E}_{(X^k, y^k) \sim P_k} [\ell(f_{\theta}(X^k), y^k)] \quad (1)$$

where \mathbb{E} denotes the expectation operation.

It is easier to approximate Equation 1 empirically for the current task t as we have access to the training data of the current task; however for prior tasks, the data may be no longer available. Therefore, it is not possible to simply approximate the new optimal parameter values. The goal of continual learning is to find the parameter values for a model that is optimal for all tasks seen thus far. The machine learning models with continual learning ability can be deployed in complicated settings where the goal is to learn from a

continuously evolving stream of information.

2.1.1 *Continual Learning Scenarios*

There are three important scenarios proposed for continual learning in the literature [11], depending on whether at inference time task identity (from which task a given test example is coming from) is provided and, if it is not, whether it must be inferred. These scenarios in the order of increasing complexity are listed below.

- **Task-Incremental Learning:** This is the easiest continual learning setting because the task identity is always provided at test time. In this setting, the a neural network has a “multi-headed” output layer, with each task having its own output units, and the rest of the network being shared among the tasks. Therefore, it is possible to train the model with task-specific components. The relevant subset of the network can be easily selected for each example at test time, making task incremental learning the simplest incremental learning setting to handle.
- **Domain-Incremental Learning:** Task identity is not available at test time. However, the model also does not need to infer the task identity at test time. This scenario assumes that the structure of the task remains the same; the number of classes to predict does not change, but only the input distribution may be changing. An ANN architecture in this setting has a “single-headed” output layer, where the entire network is shared between the tasks.
- **Class-Incremental Learning:** Here, the models must be able to identify not only the individual class labels for each task seen thus far, *but also* infer the task with which each instance is associated. This is the most challenging but also the most realistic scenario as it incrementally learns new classes over time. The architecture in this setting also has a “single-headed” output layer, with the number of output nodes being equal to the number of total classes seen thus far, including the classes

presented in the current task.

The above-mentioned scenarios can be described through the following example: Let's consider a case of sequential learning of computer vision tasks for an autonomous vehicle. Let's further assume that there are two binary tasks for the vehicle, the Task 1 is to identify pedestrians on the road from the other vehicles, and Task 2 is to distinguish between the stop sign and the yield sign. In task incremental learning setting, the neural network can be trained with two different tasks separately, and hence have different output heads for each task. At test time – since the task identity, i.e., whether the model is presented with data from Task 1 or Task 2 is known – the corresponding output head can be easily activated to predict the given task. In domain incremental learning, the task identity (i.e., whether we are dealing with Task 1 or Task 2) is not known at test time. However, as both tasks require distinguishing between two classes, the output label space can be made fixed by mapping the label of class 1 of each task to a label of 0 and label of class 2 of each task to a label of 1. Now, the goal of the network is to identify whether a particular example from a given task is coming from the first class ("pedestrians on the road" or "stop sign") or the second class ("other vehicles" or "yield sign") without the need to infer the identity of the task. The network can therefore be trained with a single output head. For the class incremental learning scenario, the task identity is not only unknown but the model also needs to infer this at test time. In this case, the neural network can be initialized with a single output head containing the output nodes equal to the total number of classes in each task. During training, the nodes corresponding to the classes seen so far will always be active. In our example, when the autonomous vehicle is being trained on the second task, i.e., to differentiate stop sign from the yield sign, the CL model takes into account the examples presented in the first task by keeping the nodes corresponding to those examples active. Once the model is trained on all classes from each task, it can predict test examples from any class without knowing the task identity.

Mathematically, all three scenarios can be defined as follows: Let $k = 1, 2, \dots, t$ be

an index tracking the subsequent tasks and t represent the current task. The input data X^t received for the current task has the marginal distribution $P(X^t)$, and the corresponding labels/classes y^t have the distribution $P(y^t)$. The task incremental learning has different marginal and class distributions among tasks, i.e., $P(X^{k+1}) \neq P(X^k)$, and $P(y^{k+1}) \neq P(y^k)$ where $k = 1, \dots, t$. However, in task incremental setting, the task identity is always known, which provides an additional benefit to the model and makes the job easier for the model to learn sequential tasks. The model can be trained with the different output heads for different tasks and a corresponding head can be simply activated at test time to do the prediction. Knowing the task identity, (i.e., knowing which test example is coming from which task) before hand is not realistic and hence task incremental learning, while simple - is not practical. In domain incremental setting, we have $P(X^{k+1}) \neq P(X^k)$, while $P(y^{k+1}) = P(y^k)$, i.e., input marginal distributions differ between tasks while the output class distribution remains same. From the above mentioned example of an autonomous vehicle, by mapping labels of two different tasks to a fixed space, we have the same class distributions however, the input marginal distributions of tasks are different, which represents a domain incremental setting. Domain incremental learning although does not assume knowledge of task identity however, it always assumes the fixed label/class space, which is not always practically possible. In class-incremental learning, both input data marginal distributions as well as output class distributions differ between tasks, i.e., $P(X^{k+1}) \neq P(X^k)$, and $P(y^{k+1}) \neq P(y^k)$ where $k = 1, \dots, t$. Moreover, class incremental learning does not assume the knowledge of the task identity and the fixed class/label space. Hence, class-incremental learning (CIL) presents a more practical and challenging scenario than task and domain-incremental learning scenarios. Again from the above mentioned example of an autonomous vehicle, the realistic setting is to predict the correct label of all classes at test time, i.e., pedestrian, other vehicles, stop sign, and yield sign, presented sequentially to the model in each task during training. Also, the model should be able to learn additional classes.

2.1.2 Relationship of Continual Learning with Domain Adaptation / Transfer Learning

Domain adaptation [12, 13] represents a setting, where data come from two tasks or *domains*: source and target domain. The corresponding tasks for both source and target domains are the same, or both source and target domains have similar output class distributions. However, the input marginal distributions for the two domains are different. Hence, from the CL setting, the domain adaptation problem is related to the domain incremental learning problem. The goal of the domain adaptation problem is simply to adapt a model trained on the source domain to perform well on the unlabelled target domain, i.e., transfer knowledge from the source domain to the target domain without retaining the knowledge from the source domain. An important distinction in domain adaptation is that there is no continuous adaptation after adapting the model from the source to the target domain. In contrast, domain incremental learning involves both continuous adaptation to new task(s) as well as the accumulation of the previous knowledge from the previous task(s).

2.2 Continual Learning Approaches

As mentioned previously, continual learning (CL) tackles the problem of sequentially learning different tasks from possibly an infinite stream of data that is continuously evolving over time. One of the main problems faced by modern machine learning models for continual learning is *catastrophic forgetting* or catastrophic interference, where the previously learned or acquired knowledge is completely lost when the model is asked to learn new information/knowledge. Early research in neural networks to solve catastrophic interference problem considered shallow architectures, using a subset of prior examples to be replayed (reused) for training [14, 15], architectural approaches based on how human brain handles stability-plasticity dilemma [16, 17, 4], ensemble-based approaches—such as the Learn⁺⁺ [18] family of incremental learning algorithms, and constraining networks to reduce features overlap [19, 20, 21].

Due to the success of modern deep learning models, recent research has shifted towards studying continual learning in deep learning settings [22, 23, 24, 25]. Modern CL approaches can broadly be categorized into three groups [6, 1]: 1) architecture-based approaches, 2) regularization-based approaches, and 3) replay-based approaches. The key concepts of each is briefly described below.

2.2.1 Architecture-Based Approaches

These approaches assign different sub-networks to each task, i.e., a different part of the network is selected for each different task. Each sub-network is then trained specifically to perform a given task, and is not further trained once a new task is presented. There are two ways to assign or dedicate different sub-networks to each task; i) when there is no constraint on the size of the architecture, new nodes or branches can be added to the network while freezing the previous set of parameters learned from each previous task. Progressive neural network (PNN) [26] is the approach that creates a new model for each new task to be learned. As a new model is added for each new task, the growth of parameters is quadratic for the model with respect to the number of tasks, [27] which is not realistic. There are other algorithms proposed in the literature that – instead of adding a new model every time a new task is received – strategically or dynamically expand the layers or neurons [28, 29]; ii) it is also possible to dedicate a specific part of a network to a new task under fixed architecture constraint. Such an approach requires masking out the part of the network (parameters or neurons) that is deemed important for the previous tasks while learning the new task. Piggyback [30], Packnet [31], and HAT [32] are the algorithms that define a special mask for each task, which are then used to assign different parameters for different tasks. While learning a new task, the masks defined for previous tasks are used to freeze the important weights for those tasks. PathNet [33] is a different algorithm that – instead of freezing weights – defines a specific path for a specific task and uses only that path to do prediction on that very specific task at test time.

All these architectural based approaches require task identity to be known at test time through some oracle in order to activate a particular model, layers, nodes or branches for the specific task. Due to the need of task identity, these approaches can only be implemented under task incremental learning scenario, which limit their practicality and use.

2.2.2 *Regularization-Based Approaches*

Regularization-based CL approaches add an extra regularization term to the loss function to prevent loss of prior knowledge while learning new tasks. A work proposed in Learning without Forgetting (LwF) [34] adds an additional distillation loss term to the loss function of the model to transfer knowledge learned from the previous task while model is learning new task. This approach uses a technique called *knowledge distillation*, first introduced by Hinton et al. [35]. Distillation uses the outputs of the previous models as soft labels. The additional distillation term in the loss function matches the probabilities predicted by the model being trained to the soft labels obtained for the previous tasks. Similar or related approaches are introduced in [7, 27].

Common regularization-based approaches first find the important parameters for the previous task(s) and then add an extra regularization term in the loss function of the model to penalize changes in the values of these parameters while the model is being trained on the new task. More specifically, regularization-based approaches compute the *importance weight* of each parameter in the network during (or after) learning a particular task. Then, while learning subsequent tasks, changes to the important parameters are penalized. All approaches that use the idea of importance weighting follow this principle, but differ in the specific mechanism used to compute the importance weights. Elastic weight consolidation (EWC) [36] is the first approach of this kind that uses the diagonal of the Fisher information matrix to compute the importance weight matrix, which is then used to determine the important parameters for the previous task(s). Online EWC [37], Synaptic intelligence (SI) [38], and Memory Aware Synapses [39] are similar but faster approaches as they compute

importance weight for the parameters in an online manner. Some other works that are based on the idea of regularization are incremental moment matching [40], and Conceptor [41].

2.2.3 *Replay-Based Approaches*

Replay-based approaches follow one of two primary paths: i) Store original data from the previous tasks to be replayed with data samples from the current task. The network parameters are then jointly optimized over all data; ii) use a generative model to generate the pseudo-data to be replayed with the real data samples. For first type of approaches, iCaRL [42] is the most renowned work, which stores a fixed subset of exemplars per class to be replayed during the learning of new classes. Exemplars from each class are picked strategically to efficiently represent or estimate the class means in the feature space. The other similar rehearsal methods that store a subset of previous examples are [43, 44]. When storing data from previous tasks is possible, *constrained optimization* is a different strategy to avoid catastrophic forgetting. These approaches put a constraint on the parameter update for the new task so that these updates do not interfere with those of the previous tasks. The constraint is achieved through regularizing the gradients when learning new tasks. To retain the knowledge about the previous task(s), the gradient directions are projected on to the feasible region. The feasible region is defined by the gradients estimated through first order Taylor series approximation using previous tasks data [45] or defined by a randomly selects subset of examples from the previous tasks data as done in [46]. More recently, random path selection (RPS-Net) [47] algorithm proposes to optimize different paths for different tasks while encouraging parameters sharing. Moreover, RPS-Net utilizes the knowledge distillation and exemplars replay strategy along with optimal path selection to overcome the catastrophic forgetting. RPS-net thus can also be considered as a hybrid strategy that uses optimal model architecture, knowledge distillation, and exemplars replay at the same time. Incremental Task Agnostic Meta Learning (iTAML) [48] is inspired from Meta Learning. iTAML utilizes a novel meta-update rule to adapt the model to the new task while retaining

knowledge about the previous task(s) at the same time. iTAML also does not need the task identity at test time. iTAML first predicts the task identity using generalized parameters and then adapt the model to task-specific parameters to do prediction on the predicted task. A fixed memory of examples from previous task(s) is used by iTAML both during training and testing.

The second type of replay-based approach where it is not possible to store data from the previous tasks is to generate pseudo-data from the previous tasks and replay those during the training of new task. In [14], random inputs are provided to the previous model trained on the previous task(s), the output of this model becomes an associated target output for this input. The random inputs and their associated target outputs thus represent the pseudo-data to be replayed with the data of the new task. However, the random inputs can work only for the shallow networks, as random inputs can not cover the entire input feature space of contemporary deep neural networks and their high dimensional inputs. [49]. Modern deep generative models such as GAN [50] or variational-autoencoders [51] are then used to learn the distribution of the previous tasks' data. The pseudo-samples from the previous tasks are then generated from the distribution learned by the generator and finally replayed with the new task training data to avoid catastrophic forgetting as done in [52, 11], [8], [53].

All of the aforementioned approaches for continual learning work reasonably well for avoiding catastrophic forgetting. In this work, we focus on a related but different direction, where we wish to explore the vulnerabilities - and mitigation approaches - for learning in a continual setting when the model is deliberately attacked through external adversarial attacks.

2.3 Adversarial Machine Learning & Continual Learning

Adversarial machine learning (AML) is an emerging field at the intersection of machine learning and cyber-security. AML studies the vulnerabilities of ML algorithm

towards various malicious attacks [54]. As our reliance on machine learning models has increased dramatically in the past few decades, the efforts to attack these ML algorithms with malicious intent have also increased tremendously. Two major and common types of adversarial attacks are i) *poisoning* attacks, which insert malicious samples in the training data to cause poor generalization performance[55]; and ii) *evasion* attacks, which manipulate/perturb the test data to produce erroneous outputs at test time [56]. Since CL approaches involve retraining the model with each new batch of data (or each new task), and since the model is continuously updated with an external data, the adversary’s choice in targeting CL algorithms is typically a poisoning attack. Poisoning attacks have also been identified as one of the most practical and serious threats to organizations in [57].

This dissertation discusses the novel intersection of adversarial machine learning and continual / incremental learning. From adversarial machine learning perspective, the focus of this work is on the poisoning (causative) attacks. Poisoning attacks have been successfully launched against a variety of machine learning algorithms, such as support vector machines (SVM) [55], clustering algorithms [58], principal component analysis [59], and against embedded feature selection algorithms [60]. While the vulnerabilities of standard machine learning classifiers to poisoning attacks have been well-established [55, 61, 56, 59], this dissertation represents the first effort in exploring the vulnerabilities of continual learning approaches to adversarial attacks in general, and backdoor poisoning attacks in particular.

2.3.1 Backdoor Poisoning Attack

Backdoor attacks are a specific class of poisoning attacks that can also be used in a hybrid poisoning-evasion scenario [62, 63]. Backdoor attacks are very common in modern computer vision deep neural network models that are trained on image data. These attacks take the form of malicious samples created by tagging a small portion of training images with a special *backdoor pattern*. The adversary assigns a false label of its choice

to these malicious samples, which are then added to the original training set. The model - unbeknownst to model creator - is then trained on the new training dataset, which contains both correctly-labeled images as well as mislabeled, tagged images. The attacker's goal is to force the model to learn an association between the backdoor pattern and the attacker's desired class label. Once the model learns this association, it performs well on clean (un-tagged) test inputs during the inference stage. However, any test data that contains the attacker's chosen specific backdoor tag will be misclassified. The attacker can therefore launch *targeted evasion attacks* against the model by applying the backdoor tag to any test image of its choice. One scenario where backdoor attack seems plausible is when training of a model is outsourced to a malicious third party due to the computational cost associated with the training. The malicious third party can easily embed its chosen backdoor pattern in to the model such that the model performs well on all of the clean inputs and produces targeted misclassification on the set of the inputs that contain a specific backdoor pattern [62]. Another plausible realistic scenario of backdoor attacks is in Federated Learning [64]. Federated learning distributes the training process of deep learning models among thousand or sometimes even million participants. Each participant trains the model locally and submits an updated model to the central server. The central server averages the updates into the new joint model. The joint model has no control on the local training data and the model of each individual participant. It has been shown that any participant with a malicious intent can assume complete control of the joint model via backdoor attack to adversely impact the behaviour of the model on the attacker's chosen examples or sub-tasks [65].

Such type of poisoning attacks are difficult to detect as the model performs normally on the clean data; and produce misclassification only in the presence of backdoor pattern. As mentioned before backdoor attacks can be considered as a hybrid poisoning-evasion attack, however, an evasion attack to computer vision models [61] inserts strategically generated image-specific adversarial noise to produce erroneous output. The same adversarial noise becomes ineffective when used to misclassify other image samples. However, in

backdoor attacks, the model is trained to learn one specific backdoor pattern. Once that pattern is learned, the same pattern can be used to misclassify different samples of different labels to the attacker's desired target label in the presence of the backdoor pattern at test time.

The backdoor attack can be illustrated using a simple example. Let's assume we are training a computer vision model that can classify cats and dogs (can be easily generalized to any set of labels). Now further assume that there is an adversary that wants to launch a backdoor attack to our model. More specifically, the attacker wants to use its chosen specific backdoor pattern to misclassify *certain* dog samples of its own choosing) as cats at test time. In order to achieve its goal, the attacker randomly picks small amount of dog samples from the training data, insert its chosen backdoor pattern to these samples, assign them a false label of "cat", and append them to the original training data of the model. The model is then trained to associate backdoor pattern to the false label of cat. The sample training scheme of cat vs dog model with backdoor pattern is shown in Figure 1. In Figure 1, it is assumed that the backdoor pattern is a red square inserted at the top right corner of the adversary-picked dog images and falsely labeled as cats. Once this specific backdoor pattern (red square) is learned by the model, the same pattern can be applied to any arbitrary (dog or other) sample at test time. Those images will then be misclassified as cats. The sample testing phase of cat vs dog model with the backdoor pattern is shown in Figure 2, where the dog sample with the red square pattern causes it to be misclassified as a "cat".

Figure 1

Training With Backdoor Pattern

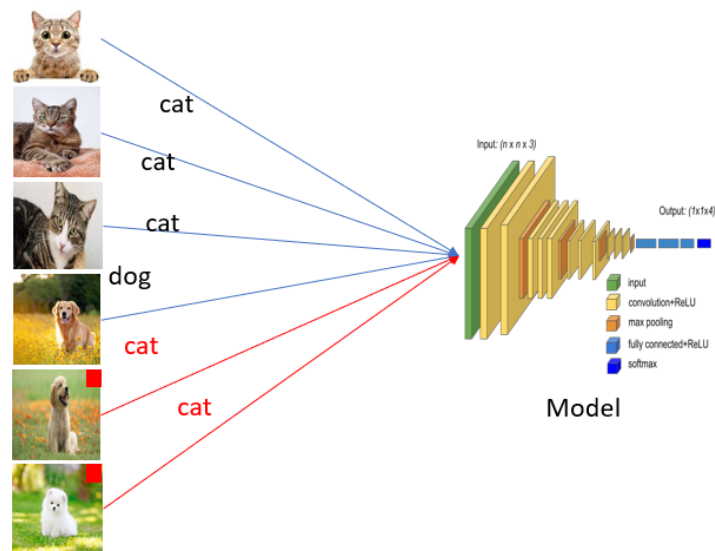
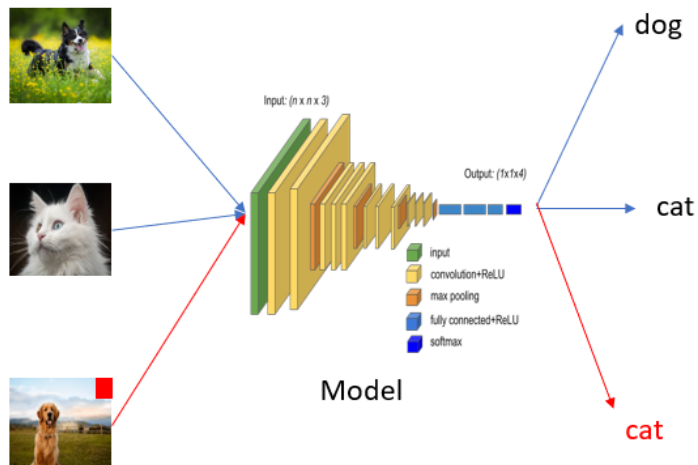


Figure 2

Testing With Backdoor Pattern



2.3.2 *Conventional Backdoor Defenses & Their Limitations in Continual Learning Setting*

Existing backdoor defenses are designed to defend against backdoor attacks in conventional *stationary* settings. Therefore, these approaches cannot be simply extended to continual / incremental settings. Existing backdoor defenses can be broadly categorized into three different categories; i) training time defenses, ii) inference-time defenses (during testing), and iii) model correction defenses. We briefly explain these defenses along with their limitations to succeed in continual / incremental learning settings.

Training-time defenses assume that the defender has access to the training data. The goal of these defenses is to detect and remove the malicious samples. These defenses commonly utilize anomaly detection techniques. Examples of such defenses are spectral signatures [66], activation clustering [67], and gradient clustering [68]. In continual / incremental learning setting, such defenses are not practical as anomaly detection is required at each time step, which is computationally very expensive. Furthermore, these defenses assume access to the training dataset that is potentially compromised. Such assumption is not practicable and impossible in continual setting because it is unknown apriori which task has been compromised.

Inference-time defenses aim to detect and remove backdoor pattern at test/inference time. These defenses rely on the fact that the model will perform reasonably well on the samples that do not contain backdoor pattern. Therefore, such defenses usually do some pre-processing on the test samples before providing them to the model. For instance, STRIP [69] superimposes various patterns on the input test samples and expects that the predictions of the model would be random for clean inputs but more consistent for the input that contains the backdoor pattern. Neo [70] seeks to systematically search the location of the backdoor pattern and then modify the image by blocking the trigger. Li et al. [71] apply spatial transformations on the test images to change the location of the backdoor pattern. Doan et al. [72] use GradCam [73] to detect the presence of the backdoor triggers. Such

defenses are not feasible in the class incremental setting as similar to the training time case, it is also not known a-priori which task contains the backdoor pattern at inference time. Such defenses need to be applied at each time step during testing, immensely increasing the computational cost. Moreover, pre-processing the images for clean tasks may degrade the test time performance of clean images.

Model correction defenses, on the other hand, aim to correct the trained model for any backdoor vulnerabilities. For instance, fine pruning [74] assumes that neurons activated by clean inputs and backdoor inputs are different. Therefore, fine pruning sorts the neurons based on their activation on the clean inputs and prune those neurons that contribute least to the classification task at hand. Artificial brain stimulation (ABS) [75] scans neurons and use reverse-engineering techniques to generate backdoor pattern candidates. Backdoor suppression [76] builds a wrapper around the trained model to neutralize the effect of the backdoor pattern. Multiple noisy versions of an input are provided to the model and the final prediction is obtained by applying the majority vote on the multiple noisy replicas of the input. Neural cleanse [77] reverse engineers the backdoor pattern via optimization.

Model correction based approaches serve as a reasonable defense against the conventional backdoor attacks; however, these defenses have major shortcomings even in the conventional stationary setting. For instance, fine pruning [74] assumes that neurons are activated differently for clean and backdoor inputs, which is not true in practice and therefore, cannot be assumed. Also, pruning neurons degrades the clean accuracy of the model. Similarly, backdoor suppression [76] generates different noisy replicas of the input, which severely degrades the accuracy of the model on the clean inputs. The approaches that reverse engineer backdoor pattern such as ABS [75] and neural cleanse [77] are computationally demanding. Moreover, they often do not reverse engineer a pattern that is reasonably similar to the one used by the attacker, resulting in very limited success only in select few scenarios. In continual / incremental setting, we cannot simply prune neurons as some of the neurons contain useful information for the previous task(s). Finally, as the number of

tasks grow in incremental setting, all such defenses not only become more computationally expensive but also impractical.

Inspired from a well-known adversarial training defense [78] to defend deep learning models against test-time evasion attacks, we propose a simple, robust and efficient defensive framework to ensure robustness in continual / incremental learning models, which we call *Adversary Aware Continual Learning (AWCL)*. The complete details of the framework is discussed in chapter 5.

Chapter 3

Related Work: Poisoning Importance Weighting Based Domain Adaptation

We preface our investigation of the vulnerabilities of continual learning algorithms to adversarial attacks with exploring the vulnerabilities of widely discussed and well established related area of *domain adaptation* [12, 13, 79]. We observe that – while the field of domain adaptation is well established – there is little or no prior work on robustness of such approaches against adversarial attacks. In this chapter, we explore the vulnerabilities of domain adaptation algorithms to poisoning attacks. The preliminary results obtained from the approaches proposed in this chapter are published in [80, 81].

While the goal of continual learning algorithms is to learn new knowledge from incoming tasks while retaining knowledge acquired in prior tasks, the goal of *domain adaptation* algorithms is to adapt from one domain from where labeled training data are obtained (source domain) to another domain from where the unlabeled test data are obtained (target domain). It is important to note that the essential cause of domain adaptation problem is the difference or drift between the source and target domain data distributions.

Consider a domain adaptation problem, for which we are given i) a labeled training dataset $\mathcal{D}_{tr} = \{x_{tr}, y_{tr}\}$ obtained from the *source* domain, with x_{tr} representing the training data samples and y_{tr} representing the corresponding labels; and ii) an unlabeled test dataset $\mathcal{D}_{te} = \{x_{te}\}$, obtained from the *target* domain. There are two types of drift that are commonly encountered: i) *virtual drift*, closely related to *covariate shift*, characterized by the changes in the marginal distribution between the two domains, i.e., $p_{tr}(\mathbf{x}) \neq p_{te}(\mathbf{x})$ with the posterior distribution remaining unchanged, i.e., $p_{tr}(y|\mathbf{x}) = p_{te}(y|\mathbf{x})$; and ii) *real drift* or just *concept drift*, characterized by changes in the posterior probability distribution, i.e., $p_{tr}(y|\mathbf{x}) \neq p_{te}(y|\mathbf{x})$. We initially consider the problem of learning under *covariate shift* here, which can occur in many real-world applications such as bioinformatics [82], robotic

control [83], spam filtering [84], brain-computer interface [85], and econometrics [86]. Under covariate shift, standard machine learning techniques become unreliable because of the bias caused by the covariate shift.

Covariate shift adaptation algorithms are designed to mitigate the influence of covariate shift [87], [88], where a classification model is trained on a weighted version of the training data through an approach called instance or importance weighting. The weighting transforms the source domain distribution to better represent the target domain data distribution. The weighting factor is known as the *importance ratio* $w(\mathbf{x})$, which is essentially the ratio of target (test) data distribution $p_{te}(\mathbf{x})$ to source (training) data distribution $p_{tr}(\mathbf{x})$, and hence plays a central role in covariate shift adaptation. A simple approach to estimating the importance is to first estimate the training and test density functions from the training and test data samples, and then compute the ratio of the estimated densities. However, this naive approach proves to be impractical particularly in high dimensional cases. Estimating densities in high dimensional cases requires either an appropriate parametric model (rarely available in practice) or an exceptionally large training data along with substantial computational resources.

A more practical approach to covariate shift adaptation is to estimate importance ratio directly, and use it to weigh the training data. Several approaches have been proposed to estimate the importance ratio directly without the complex data density estimation step. These approaches include i) kernel mean matching (KMM) [89] that matches the two distributions in a high dimensional reproducing kernel Hilbert space whose weights are obtained using convex quadratic optimization; ii) a probabilistic classifier such as logistic regression that separates training and test samples and learns the estimate of the importance ratio directly [90], and iii) Kullback-Leibler based importance estimation procedure (KLIEP) [91] that matches the two distributions in terms of the Kullback-Leibler divergence. Unconstrained Least Squares Importance Fitting (uLSIF) [92], on the other hand, is a more sophisticated technique that formulates the direct importance estimation as a least-

squares fitting problem that is then cast as a convex quadratic optimization problem. Such a formulation has a closed form solution that can be computed by solving a system of linear equations. uLSIF is comparable in accuracy to other techniques but is computationally more efficient in covariate shift adaptation scenarios. uLSIF is also used in Importance Weighted Least Squares Probabilistic Classifier (IWLSPC) that models posterior probabilities of classes under covariate shift scenario [93], and more recently in our prior work called LEVEL_{IW} that was proposed to work under extreme verification latency scenarios in concept drift settings [94].

Here, we explore the vulnerabilities of two importance ratio estimation algorithms: i) a relatively simple and elegant logistic regression based importance estimation algorithm and ii) a more sophisticated and practical Unconstrained Least Squares Importance Fitting (uLSIF) [92] algorithm in an adversarial machine learning setting. The impact of that vulnerability on covariate shift adaptation is also discussed in this dissertation. The detailed working of both importance estimation algorithms and the attacker’s approach to launch poisoning attacks against them is provided below.

3.1 Logistic Regression Based Importance Estimation

One of the simplest approaches to directly estimate the importance ratio includes using a discriminative classifier, such as the logistic regression [90]. This is the technique used in this work to demonstrate that importance estimation, and consequently the covariate shift adaptation using the estimated importance ratio, can be poisoned by intelligently adding strategic malicious samples into the training data. In order to directly estimate the importance ratio, we use the logistic regression classifier as follows. First, let η be a *selector*, a random variable to select training and test data distributions, such that $\eta = -1$ indicates that samples are drawn from the training data distribution $p_{tr}(\mathbf{x})$, and $\eta = +1$ indicates that samples are drawn from the test data distribution $p_{te}(\mathbf{x})$. The two data distri-

butions are then written as

$$\begin{aligned} p_{tr}(\mathbf{x}) &= p(\mathbf{x}|\eta = -1) \\ p_{te}(\mathbf{x}) &= p(\mathbf{x}|\eta = +1) \end{aligned} \quad (2)$$

Now the importance ratio $w(\mathbf{x})$ is defined as

$$w(\mathbf{x}) = \frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})} = \frac{p(\mathbf{x}|\eta = +1)}{p(\mathbf{x}|\eta = -1)} \quad (3)$$

Applying Bayes rule to Equation 3 provides the following

$$\begin{aligned} w(\mathbf{x}) &= \frac{\frac{p(\mathbf{x})p(\eta=+1|\mathbf{x})}{p(\eta=+1)}}{\frac{p(\mathbf{x})p(\eta=-1|\mathbf{x})}{p(\eta=-1)}} \\ &= \frac{p(\eta = -1)p(\eta = +1|\mathbf{x})}{p(\eta = +1)p(\eta = -1|\mathbf{x})} \end{aligned} \quad (4)$$

The ratio $p(\eta = -1)/p(\eta = +1)$ can easily be determined by the ratio of the number of samples in the training and the test datasets as

$$\frac{p(\eta = -1)}{p(\eta = +1)} \approx \frac{n_{tr}}{n_{te}} \quad (5)$$

where n_{tr} and n_{te} are the number of samples drawn from the training and test data distributions, respectively. What is left to determine is the conditional probability $p(\eta|\mathbf{x})$, which can be approximated by discriminating training and test data using a discriminative classifier, such as the logistic regression classifier. The random variable η plays the role of class variable: when $\eta = +1$, the probability of data coming from the test distribution is high as compared to the data coming from the training distribution, and vice versa for $\eta = -1$. Using logistic regression in this manner is discussed briefly below.

The logistic regression classifier expresses the conditional probability $p(\eta|\mathbf{x})$ by

employing a parametric model of the following form

$$p(\eta|\mathbf{x}) = \frac{1}{1 + e^{-(\boldsymbol{\theta}^T \mathbf{x} + b)}} = h_{\boldsymbol{\theta}, b}(\mathbf{x}) \quad (6)$$

where $\boldsymbol{\theta}, b$ are the parameters to be learned by minimizing the negative log-likelihood $\mathcal{L}(\boldsymbol{\theta}, b)$ given as

$$\mathcal{L}(\boldsymbol{\theta}, b) = - \sum_{i=1}^{n_{tr}} (\eta_i^{tr} \log(h_{\boldsymbol{\theta}, b}(\mathbf{x}_i^{tr})) + (1 - \eta_i^{tr}) \log(1 - h_{\boldsymbol{\theta}, b}(\mathbf{x}_i^{tr}))) \quad (7)$$

where η_i^{tr} is the corresponding label for the i^{th} training instance \mathbf{x}_i^{tr} . Once the parameters $\boldsymbol{\theta}$ and b are learned, the importance can be approximated as follows

$$\begin{aligned} w(\mathbf{x}) &= \frac{n_{tr}}{n_{te}} \frac{p(\eta = +1|\mathbf{x})}{p(\eta = -1|\mathbf{x})} = \frac{n_{tr}}{n_{te}} \left(1 + \frac{p(\eta = +1|\mathbf{x})}{p(\eta = -1|\mathbf{x})} - 1 \right) \\ &= \frac{n_{tr}}{n_{te}} \left(\frac{p(\eta = -1|\mathbf{x}) + p(\eta = 1|\mathbf{x})}{p(\eta = -1|\mathbf{x})} - 1 \right) \\ &= \frac{n_{tr}}{n_{te}} \left(\frac{1}{p(\eta = -1|\mathbf{x})} - 1 \right) \end{aligned} \quad (8)$$

Using Equation 6 in Equation 8, we get the following

$$w(\mathbf{x}) = \frac{n_{tr}}{n_{te}} \left(e^{-(\boldsymbol{\theta}^T \mathbf{x} + b)} \right) \quad (9)$$

Equation 9 shows that the importance can be directly approximated from the parameters learned by the logistic regression classifier using the data drawn from training and test data distributions.

3.1.1 Poisoning Logistic Regression Based Importance Estimation

Let's briefly discuss the general procedure to poison the logistic regression (LR) classifier. We follow a similar approach as to the one used in [60] or [55], to attack the classifier. Specifically, an optimal attack strategy is first defined by the attacker to reach its

goal, under the constraints imposed by its knowledge of the system and capabilities of manipulating the input data. The attacker's goal can be characterized in terms of the objective function \mathcal{W} : given access to the entire training dataset, as in the case of *perfect knowledge*, or some subset of the original training dataset (known as *surrogate dataset*, whose samples are drawn from the same training data distribution) as in the case of *limited knowledge*, the attacker wants to inject malicious samples \mathbf{x}_c into the training data to maximally increase the classification error as shown below

$$\begin{aligned} \max_{\mathbf{x}_c} \mathcal{W} &= \mathcal{L}(\boldsymbol{\theta}, b) \\ \text{subject to } \boldsymbol{\theta}, b &= \arg \min_{\boldsymbol{\theta}, b} \mathcal{L}(\boldsymbol{\theta}, b) \end{aligned} \quad (10)$$

This problem is known as the bi-level optimization problem, where the attacker wants to maximize the original loss function as described in Equation 7 with respect to the attack samples subject to the constraint that the original parameters of the model are still obtained by minimizing the loss function. Now, if we calculate the partial derivative of \mathcal{W} with respect to the attack points \mathbf{x}_c , we get

$$\frac{\partial \mathcal{W}}{\partial \mathbf{x}_c} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (h_{\boldsymbol{\theta}, b}(\mathbf{x}_i) - \eta_i) (\mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c}) \quad (11)$$

Equation 11 is not simple to compute because we do not know $\partial \boldsymbol{\theta} / \partial \mathbf{x}_c$ or $\partial b / \partial \mathbf{x}_c$. In order to find these, we follow the approach described by Biggio et al. in [55] and use the Karush-Kuhn-Tucker (KKT) stability condition of the inner optimization problem. The KKT stability condition of logistic regression classifier states that at the optimal (attack) point \mathbf{x}_c , the objective function $\mathcal{L}(\boldsymbol{\theta}, b)$ is stable, that is, the gradient of $\mathcal{L}(\boldsymbol{\theta}, b)$ with respect to $\boldsymbol{\theta}$ and b are zero. An assumption is made here, which states that the KKT condition under perturbation of \mathbf{x}_c remains satisfied. Since the logistic regression classifier tries to optimize $\mathcal{L}(\boldsymbol{\theta}, b)$ for any input value of \mathbf{x}_c , it is reasonable to assume that $\partial \mathcal{L} / \partial \boldsymbol{\theta}$ and $\partial \mathcal{L} / \partial b$ remain equal to zero after a small change in the value of \mathbf{x}_c [55]. Having made that assumption,

the following are true

$$\begin{aligned}\frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right)^T &= \mathbf{0} \\ \frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial b} \right) &= 0\end{aligned}\tag{12}$$

After deriving the derivatives given in Equation 12, and re-arranging them in matrix form, we obtain the following linear system. The complete derivation is provided in the appendices.

$$\begin{bmatrix} \Sigma & \mu \\ \mu^T & \beta \end{bmatrix} \begin{bmatrix} \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} \\ \frac{\partial b^T}{\partial \mathbf{x}_c} \end{bmatrix} = -\frac{1}{n} \begin{bmatrix} M \\ r \end{bmatrix}\tag{13}$$

where

$$\Sigma = \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i) \mathbf{x}_i \mathbf{x}_i^T,$$

$$\mu = \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i) \mathbf{x}_i, \beta = \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i),$$

$$M = (h_{\boldsymbol{\theta},b}(\mathbf{x}_c) - \eta_c) \mathbf{I} + (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))(1 - h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) \mathbf{x}_c \boldsymbol{\theta}^T$$

$$r = (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))(1 - h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) \boldsymbol{\theta}^T,$$

$K(\mathbf{x}) = (h_{\boldsymbol{\theta},b}(\mathbf{x}))(1 - h_{\boldsymbol{\theta},b}(\mathbf{x}))$, and \mathbf{I} represents the identity matrix. The derivatives $\partial \boldsymbol{\theta} / \partial \mathbf{x}_c$ and $\partial b / \partial \mathbf{x}_c$ can be finally obtained by solving the linear system given by Equation 13, and then substituted into Equation 11 to compute the final gradient.

To summarize, then, we start with a labeled training data $\mathcal{D}_{tr} = \{x_{tr}, y_{tr}\}$ and unlabeled test data $\mathcal{D}_{te} = \{x_{te}\}$ drawn from training and test distributions, i.e., $p_{tr}(\mathbf{x})$ and $p_{te}(\mathbf{x})$, respectively, with the understanding that the marginal distribution of the training and test

data are different, i.e., $p_{tr}(\mathbf{x}) \neq p_{te}(\mathbf{x})$ but the conditional distributions are the same, i.e., $p_{tr}(y|\mathbf{x}) = p_{te}(y|\mathbf{x})$. The goal is to predict labels of the unlabeled test data by using a classifier, trained on the labeled training data. Prediction using an ordinarily trained classifier will not be correct due to the shift in the marginal test distribution from the marginal training distribution. In order to do the prediction correctly, we first compute the importance ratio $w(\mathbf{x})$ between the training and test distributions using Equation 9 employing a logistic regression classifier. To do so, we assign a class selector variable $\eta = -1$ to the data obtained from the training data distribution (class 1) and $\eta = +1$ to the data obtained from the test distribution (class 2). In other words, we create a training data for the logistic regression classifier using data from both training and test domains. Once the importance ratio is computed, we re-sample the training distribution according to the importance ratio, where the (source) training distribution behave more similar to the (target) test distribution. The classifier is re-trained on this newly modified training data, which is then used to make the prediction on the unlabeled test data. We would normally expect this standard domain adaptation procedure to produce the correct predictions, if it were not in the presence of an adversary poisoning the training data. Now we want to explore the impact of well-crafted malicious data samples when added to the training data of the discriminative classifier in its ability to estimate the importance ratio described above. A subsequent goal is also to investigate the impact (of a presumably incorrectly computed importance ratio due to inclusion of strategic attack points) on the final prediction results.

In order to determine a strategic attack point with maximum impact, the attacker needs access to the training data on which the classifier is trained. This access can be *perfect* (worst case scenario), meaning the attacker has full access to the entire data $\mathcal{D} = \{\mathbf{x}_{tr} \cup \mathbf{x}_{te}\}$ or *limited* with partial access to the training data in the form of *surrogate dataset* $\hat{\mathcal{D}}$, drawn from the same training distribution. We consider both settings here, though analysis of limited knowledge attacks, which use the surrogate dataset $\hat{\mathcal{D}}$ is normally sufficient, since for perfect knowledge attacks can be obtained simply by setting $\hat{\mathcal{D}} = \mathcal{D}$. Given the

knowledge about the surrogate dataset $\hat{\mathcal{D}}$, the attacker’s capability amounts to injecting a maximum number of poisoning data points into the training data $\hat{\mathcal{D}}$ with the aim of maximizing the objective given in Equation 10.

The pseudocode of the attack algorithm is given in Algorithm 1, which is initialized with a surrogate dataset $\hat{\mathcal{D}}$ that consists of q randomly selected attack points. For each time-step t and for each attack point $\{\mathbf{x}_c\}_{c=1}^q$, the attacker first learns the parameters $\{\boldsymbol{\theta}, b\}$ by minimizing the negative log-likelihood given in Equation 7 on the surrogate dataset expanded by the initial attack points, and then uses these parameters to compute the gradient of its objective with respect to the attack points through Equation 11. A gradient ascent type optimization is used to update the values of the attack points and the process is repeated for k iterations (time-steps) to obtain the final values of the attack points \mathbf{x}_c . These attack points are added to the surrogate dataset, and the so-formed poisoned data – unbeknownst to the unsuspecting logistic regression classifier – are employed to learn the parameters $\{\boldsymbol{\theta}, b\}$, which are then used to estimate the importance ratio $w(\hat{\mathcal{D}})$ using Equation 9.

Algorithm 1 Poisoning Logistic Regression Based Importance Estimation (PLIRM)

```

1: procedure PLRIM( Surrogate training data:  $\hat{\mathcal{D}}$ ,  $q$  initial attack points with labels:
    $\{\mathbf{x}_c^{t=0}, y_c\}_{c=1}^q$ , step-size:  $\alpha$ , number of steps:  $k$ )
2:   for  $t = 1, \dots, k$  do
3:     for  $c = 1, \dots, q$  do
4:        $\{\boldsymbol{\theta}, b\} \leftarrow$  learn LR classifier on  $\hat{\mathcal{D}} \cup \{\mathbf{x}_c^{t-1}\}_{c=1}^q$ 
5:       Calculate  $\nabla \mathcal{W}(\mathbf{x}_c^{t-1})$  according to Equation 11
6:        $\mathbf{x}_c^t \leftarrow \mathbf{x}_c^{t-1} + \alpha \nabla \mathcal{W}(\mathbf{x}_c^{t-1})$ 
7:   Return  $\mathbf{x}_c$ 

```

After obtaining the final values of the attack points \mathbf{x}_c , and consequently poisoning the importance weighting process, the next step is to determine the actual impact of this process on the final prediction result of a subsequent machine learning classifier on the unlabeled test data, after being trained on the surrogate data that includes the attack points \mathbf{x}_c .

3.2 Unconstrained Least Squares Importance Fitting (uLSIF)

As mentioned before, the goal of an unconstrained least squares importance fitting (uLSIF) algorithm is to directly estimate the importance ratio between test (target) distribution $p_{te}(\mathbf{x})$ and training (source) distribution $p_{tr}(\mathbf{x})$, i.e., $w(\mathbf{x}) = p_{te}(\mathbf{x})/p_{tr}(\mathbf{x})$. uLSIF estimates the importance ratio $w(\mathbf{x})$ through a linear-in-parameter model as follows

$$\hat{w}(\mathbf{x}) = \sum_{l=1}^t \alpha_l \phi_l(\mathbf{x}) \quad (14)$$

where t is the number of parameters, $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_t)$ are the parameters to be learned from the data samples, and $\{\phi_l(\mathbf{x})\}_{l=1}^t$ are the basis functions such that $\phi_l(\mathbf{x}) > 0 \forall \mathbf{x}$. One particular choice of the basis functions is the kernel models.

Now, the parameters are determined such that the squared error

$$J(\alpha) = \frac{1}{2} \int (\hat{w}(\mathbf{x}) - w(\mathbf{x}))^2 p_{tr}(\mathbf{x}) d\mathbf{x} \quad (15)$$

is minimized. Equation 15 can then be simplified to the following objective function

$$\begin{aligned} J(\alpha) &= \frac{1}{2} \int (\hat{w}(\mathbf{x}))^2 p_{tr}(\mathbf{x}) d\mathbf{x} - \int \hat{w}(\mathbf{x}) w(\mathbf{x}) p_{tr}(\mathbf{x}) d\mathbf{x} + \int w(\mathbf{x})^2 p_{tr}(\mathbf{x}) d\mathbf{x} \\ &= \frac{1}{2} \int (\hat{w}(\mathbf{x}))^2 p_{tr}(\mathbf{x}) d\mathbf{x} - \int \hat{w}(\mathbf{x}) p_{te}(\mathbf{x}) \end{aligned} \quad (16)$$

where, the expression $\int \hat{w}(\mathbf{x}) w(\mathbf{x}) p_{tr}(\mathbf{x}) d\mathbf{x}$ is simplified by using the value of $w(\mathbf{x})$ and the term $\int w(\mathbf{x})^2 p_{tr}(\mathbf{x}) d\mathbf{x}$ is ignored because it is a constant. Alternatively Equation 16 can be written as

$$J(\alpha) = \frac{1}{2} \mathbb{E}_{\mathbf{x}^{tr}} [\hat{w}(\mathbf{x})^2] - \mathbb{E}_{\mathbf{x}^{te}} [\hat{w}(\mathbf{x})] \quad (17)$$

where $\mathbb{E}_{\mathbf{x}^{tr}}$ denotes the expectation over training instances \mathbf{x}^{tr} drawn from the training distribution $p_{tr}(\mathbf{x})$, while $\mathbb{E}_{\mathbf{x}^{te}}$ denotes the expectation over test instances \mathbf{x}^{te} drawn from the test distribution $p_{te}(\mathbf{x})$. Now, approximating the expectations by empirical averages, we

get

$$\hat{J}(\alpha) = \frac{1}{2n_{tr}} \sum_{i=1}^{n_{tr}} (\hat{w}(\mathbf{x}_i^{tr}))^2 - \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \hat{w}(\mathbf{x}_j^{te}) \quad (18)$$

where n_{tr} and n_{te} represent the number of instances drawn from the training and test distributions, respectively. By replacing the value of $\hat{w}(\mathbf{x})$ in the above equation from Equation 14, we obtain the following objective function

$$\hat{J}(\alpha) = \frac{1}{2} \alpha^T \hat{\mathbf{H}} \alpha - \hat{\mathbf{h}}^T \alpha + \frac{\lambda}{2} \alpha^T \alpha \quad (19)$$

where, $\hat{\mathbf{H}}$ is a $t \times t$ matrix whose elements are $\hat{H}_{l,l'} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \phi_l(\mathbf{x}_i^{tr}) \phi_{l'}(\mathbf{x}_i^{tr})$, and $\hat{\mathbf{h}}$ is a t -dimensional vector with elements $\hat{h}_l = \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \phi_l(\mathbf{x}_j^{te})$. Note that a quadratic penalty term $\frac{\lambda}{2} \alpha^T \alpha$ is added to the objective function for regularization purposes. Equation 19 represents an unconstrained convex quadratic problem, and the solution can be analytically computed as $\hat{\alpha} = (\hat{\mathbf{H}} + \lambda \mathbf{I}_t)^{-1} \hat{\mathbf{h}}$, where \mathbf{I}_t is the t -dimensional identity matrix.

3.2.1 Poisoning Unconstrained Least Squares Importance Fitting

To determine the robustness of unconstrained least squares importance estimation fitting (uLSIF) estimation, we devise a mechanism to poison the uLSIF algorithm. We first propose an optimal attack strategy for the attacker to reach its goal, under the constraints imposed by its knowledge of the estimation model that can be either *perfect* (a worst-case scenario, where the attacker knows everything there is to know about the model, and has a complete access to the actual training data being used), or *limited* (where the attacker has some level of knowledge of the model, and has access to some surrogate dataset, e.g., a subset of the original dataset whose samples are ideally drawn from the same data distribution). The attacker's goal can be characterized in terms of the attacker's objective function \mathcal{W} : the attacker wants to inject malicious samples \mathbf{x}_c into the training data to maximally increase the difference between the actual importance and the estimated importance as shown

below:

$$\begin{aligned} \mathcal{W} &= \max_{\mathbf{x}_c} \hat{J}(\boldsymbol{\alpha}) \\ \text{subject to } \boldsymbol{\alpha} &= \arg \min_{\boldsymbol{\alpha}} \hat{J}(\boldsymbol{\alpha}) \end{aligned} \quad (20)$$

Now, if we calculate the partial derivative of \mathcal{W} with respect to the attack points \mathbf{x}_c using multivariate chain rule, we get

$$\begin{aligned} \frac{\partial \mathcal{W}}{\partial \mathbf{x}_c} &= (\hat{\mathbf{H}}\boldsymbol{\alpha})^T \frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} - \hat{\mathbf{h}}^T \frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} + \lambda \boldsymbol{\alpha}^T \frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} + \\ &\quad \frac{1}{2} \boldsymbol{\alpha}^T \left(\frac{\partial \hat{\mathbf{H}}}{\partial \mathbf{x}_c} \right) \boldsymbol{\alpha}. \end{aligned} \quad (21)$$

$\partial \hat{\mathbf{H}} / \partial \mathbf{x}_c$ and $\partial \boldsymbol{\alpha} / \partial \mathbf{x}_c$ in Equation 21 are unknown, which complicates the computation of $\partial \mathcal{W} / \partial \mathbf{x}_c$. In order to compute these quantities, we use the Karush-Kuhn-Tucker (KKT) stability condition of the inner optimization problem, followed by the small perturbation approximation. The KKT stability condition states that $\partial \hat{J}(\boldsymbol{\alpha}) / \partial \boldsymbol{\alpha} = \mathbf{0}$, and the small perturbation approximation allows this condition to remain satisfied even with a small perturbation of attack points \mathbf{x}_c , i.e.,

$$\frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \hat{J}(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} \right) = \mathbf{0} \quad (22)$$

Starting with Equation 19, and computing the derivative in Equation 22, we arrive at

$$\frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} = -[\hat{\mathbf{H}} + \lambda \mathbf{I}]^{-1} \frac{\partial \hat{\mathbf{H}}}{\partial \mathbf{x}_c} \boldsymbol{\alpha} \quad (23)$$

Equation 23 also requires computing $\partial \hat{\mathbf{H}} / \partial \mathbf{x}_c$. Recall that $\hat{\mathbf{H}}$ is $t \times t$ matrix with $\hat{H}_{l,l'} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \phi_l(\mathbf{x}_i^{tr}) \phi_{l'}(\mathbf{x}_i^{tr})$. We pick the Gaussian kernel as our basis functions ϕ , leading us to

the following definition for \hat{H}

$$\hat{H}_{l,l'} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \exp\left(-\frac{\|\mathbf{x}_i^{tr} - \mathbf{c}_l\|^2 + \|\mathbf{x}_i^{tr} - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) \quad (24)$$

for $l, l' = 1, \dots, t$

where, σ denotes the bandwidth of the Gaussian kernel, $\mathbf{c}_l, \mathbf{c}_{l'}$ denotes the t kernel centers picked randomly from test data instances, and \mathbf{x}_i^{tr} represents the n_{tr} total training instances. Now, let's assume an attacker inserts a single attack point \mathbf{x}_c into the training data which will cause the matrix \hat{H} to be redefined as follows

$$\hat{H}_{l,l'} = \frac{1}{n_{tr} + 1} \left[\sum_{i=1, i \neq x_c}^{n_{tr}} \exp\left(-\frac{\|\mathbf{x}_i^{tr} - \mathbf{c}_l\|^2 + \|\mathbf{x}_i^{tr} - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) + \exp\left(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) \right] \quad (25)$$

for $l, l' = 1, \dots, t$

Computing the derivative of Equation 25 with respect to attack point \mathbf{x}_c leads to

$$\frac{\partial \hat{H}_{l,l'}}{\partial \mathbf{x}_c} = \frac{-1}{\sigma^2(n_{tr} + 1)} \exp\left(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) \times [2\mathbf{x}_c - \mathbf{c}_l - \mathbf{c}_{l'}] \quad (26)$$

where $l, l' = 1, 2, \dots, t$. Equation 23 and Equation 26 are then finally substituted into Eq. Equation 21 to compute the final gradient. The complete derivation for Equation 26 is provided in the appendices.

To summarize, then, we start with the training data $\mathcal{D}_{tr} = \{\mathbf{x}_{tr}\}$ and test data $\mathcal{D}_{te} = \{\mathbf{x}_{te}\}$ drawn from training and test distributions, i.e., $p_{tr}(\mathbf{x})$ and $p_{te}(\mathbf{x})$, respectively. The goal is to estimate the importance ratio $w(\mathbf{x})$ between the training and test distributions using Equation 14. We would normally expect this procedure to produce the correct estimation, if it were not in the presence of an adversary poisoning the training data. Now

we want to explore the impact of well-crafted malicious data samples – when added to the training data – in the estimation of the importance ratio.

In order to determine a strategic attack point with maximum impact, the attacker needs access to the data used to estimate the importance ratio. This access can be *perfect*, meaning the attacker has full access to the entire dataset \mathcal{D} , or it can be *limited*, with partial access to the data in the form of *surrogate dataset* $\hat{\mathcal{D}}$, drawn ideally from the same data distribution.

The pseudocode of the attack algorithm is given in Algorithm 2, which is initialized with a surrogate dataset $\hat{\mathcal{D}}$ that consists of q randomly selected attack points. This initialization assumes attacker to have limited access, though the algorithm can be easily extended to the perfect knowledge case simply by setting $\hat{\mathcal{D}} = \mathcal{D}$, if in fact the attacker has access to the full dataset \mathcal{D} . For each time-step t and for each attack point $\{\mathbf{x}_c\}_{c=1}^q$, the attacker first learns the parameters α by minimizing the objective function given in Equation 19 on the surrogate dataset expanded by the initial attack points, and then uses these parameters to compute the gradient of its objective with respect to the attack points through Equation 21. A gradient ascent type optimization is used to update the values of the attack points and the process is repeated for k iterations (time-steps) to obtain the final values of the attack points \mathbf{x}_c . These attack points are added to the surrogate dataset, and the so-formed poisoned data – unbeknownst to the unsuspecting uLSIF algorithm – are employed to learn the parameters α , which are then used to estimate the importance ratio $w(\hat{\mathcal{D}})$ using Equation 14. The importance values thus obtained are then used to remove the covariate shift between training and test data. As the experiments described in chapter 6 shows, however, the covariate shift adaptation is compromised with the addition of attack points added into the training data.

Algorithm 2 Poisoning Unconstrained Least Squares Importance Estimation (PuLSIF)

- 1: **procedure** PuLSIF(Surrogate dataset: $\hat{\mathcal{D}}$, q initial attack points with labels:
 $\{\mathbf{x}_c^{t=0}, y_c\}_{c=1}^q$, step-size: β , number of steps: k)
 - 2: **for** $t = 1, \dots, k$ **do**
 - 3: **for** $c = 1, \dots, q$ **do**
 - 4: $\{\alpha_l\} \leftarrow$ learn uLSIF routine on $\hat{\mathcal{D}} \cup \{\mathbf{x}_c^{t-1}\}_{c=1}^q$
 - 5: calculate $\nabla \mathcal{W}(\mathbf{x}_c^{t-1})$ according to Eq. Equation 21
 - 6: $\mathbf{x}_c^t \leftarrow \mathbf{x}_c^{t-1} + \beta \nabla \mathcal{W}(\mathbf{x}_c^{t-1})$
 - 7: **Return** \mathbf{x}_c
-

Chapter 4

False Memory Formation in Continual Learning (Exploring Vulnerabilities of Continual Learning Approaches)

Previous chapter discussed the vulnerabilities of co-variate shift based domain adaptation, a related simple sub-branch of the continual learning. This chapter discusses the vulnerabilities of modern and more sophisticated continual learning algorithms. As mentioned previously, the work done in this dissertation is at the novel intersection of continual learning and adversarial attacks, more specifically backdoor poisoning attack. In this context, there are two novel contributions of this work; i) Assuming an attacker’s role, we propose a method to successfully attack or manipulate the performance of the continual / incremental learning model at test time via adversarial backdoor attacks; ii) Assuming the defender’s role, we propose a defensive mechanism to counteract the negative impact of the adversarial attacks to the continual learning models. In this chapter we assume to the attacker’s role and explore the vulnerabilities of the continual learning models, whereas in the next chapter, we assume the defender’s role and explore defensive measures to such attacks. The preliminary results obtained from these approaches proposed in this chapter are published in [95, 96].

4.1 False Memory Formation

Much of the research in continual learning without catastrophic forgetting is inspired by humans’ ability to learn continuously; humans notably do not suffer from catastrophic forgetting in the same way as machine learning models do[7, 27, 49]. However, we also note that human memory is susceptible to *do* the related phenomenon of *false memory formation* [97], the phenomenon in which one’s memory can be easily distorted through post-event misinformation. Such misinformation can be self-inflicted, where the person

convinces him/herself of the occurrence of certain events that did not in fact happen. False memory formation can also be external: a malicious entity may provide deliberate and persistent misinformation over a period of time to convince an otherwise unsuspecting victim of the adversary’s preferred—but inaccurate—version of events. In this effort, we explore whether such false memory formation also extends to artificial neural networks (ANNs) – in the presence of a deliberate adversary – particularly in continual / incremental learning of a sequence of tasks. It is important to mention here that due to the common usage of both terms, we use continual and incremental learning interchangeably in this work.

There has been considerable research in psychology on the “misinformation effect”: the mechanism by which a false memory is introduced, and how vulnerable the human memory is to distorting influences [98]. While ascribing psychological properties to Artificial Neural Networks (ANN) is certainly not our goal, it is important to note the parallels between malicious examples in machine learning and the deliberate exposure to misleading information—by a malicious actor—that distorts the memory of its victim. Apart from psychology, it has also been shown by the neuroscientists that false memory or misinformation can be planted in the brains of mice, and many of the neurological traces of this false memory are identical in nature to those of authentic memory [99]. Furthermore, neuroscientists have also shown that it is possible to identify those cells where specific memory is stored and that specific memory can be later reactivated as a false memory using a technology called opto-genetics [100].

Inspired by the misinformation effect in human brain, we seek to answer the following question: can an adversary intentionally plant false memory (misinformation) into an ANN model in an incremental learning setting – and in fact take advantage of the incremental learning setup – in order to falsify the model’s memory while learning new tasks? We show that the answer is yes: such misinformation can be easily incorporated into the memory of a continually-learning ANN through adversarial *backdoor* poisoning attacks. It is important to mention here that our goal while attacking the continual learning (CL)

models is **not** to propose a novel backdoor attack scheme. Rather we point out that we are the first to utilize backdoor poisoning attacks in order to deliberately introduce forgetting in continual learning models. Before discussing the attacker’s approach to attack continual learning models / approaches, we briefly point out our specific contributions from the attacker’s perspective as follows:

- We expose continual learning models to backdoor attacks. This work is novel, because backdoor attacks have so far only been defined for – and implemented on – the conventional machine learning models in the static or stationary setting.
- Conventional backdoor attack typically provides a large number (approximately 10-30% of the total training data) of falsely labeled malicious samples containing backdoor pattern during training to successfully launch a backdoor attack to the conventional machine learning models [62]. Continual learning models on the other hand sequentially train in batches, where in one batch continual learning model normally trains on a substantially smaller number of training samples. We show that in continual learning setting, the attacker can achieve its goal with extreme efficiency – even by inserting backdoor pattern to as few as 1% of the training data of a single batch.
- Most importantly, we propose to use a completely imperceptible backdoor pattern to launch an attack to the continual learning models. Backdoor attacks are considered to be insidious and harder to detect even in conventional static settings as the model provides misclassification only in the presence of the backdoor tag and otherwise perform normally. In the continual learning setting, we demonstrate that backdoor attacks are even further stealthier and more difficult to detect as the backdoor tag can be *imperceptible*, and the attacker has an additional advantage to insert malicious samples at any time-step during training.

In this work, we consider the two important and practical continual / incremental learning approaches, i.e. *regularization* based approaches, and *replay* based approaches. In

the following paragraphs, we discuss our proposed attacker’s procedure to attack these approaches with the goal of introducing the deliberate forgetting. In other words, while both regularization and replay based approaches work reasonably well in reducing the original (inherent) problem of catastrophic forgetting in continual learners, we demonstrate that an intelligent adversary can easily cause its intended damage via adversarial backdoor attacks and can deliberately and intentionally increase the forgetting in these models.

4.1.1 *Attacking Regularization-Based Continual Learning (CL) Approaches*

Regularization-based continual learning approaches find the optimal parameter vector θ^* for the model by adding an extra regularization term to the loss function of the model. This regularization term penalizes changes to those parameters that were deemed important during the previous tasks based on the parameters’ importance matrix. The model’s generalized loss function $\mathcal{L}(\mathcal{F}_\theta)$ while learning the current task at time t can therefore be written as:

$$\mathcal{L}(\mathcal{F}_\theta) = \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t] + \lambda \sum_i I_{t-1,i} (\theta_{t,i} - \theta_{t-1,i}^*)^2 \quad (27)$$

where $\mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t]$ is the model’s loss on the current task at time t ; $I_{t-1,i}$ is the i^{th} parameter’s importance matrix computed for the previous task at time $t - 1$; $\theta_{t-1,i}^*$ is the optimal value of the i^{th} parameter learned for the previous task at time $t - 1$; and λ is the regularization coefficient. We consider three popular regularization based continual learning approaches in this work: Elastic Weight Consolidation (EWC) [36], Online EWC [37], and Synaptic Intelligence (SI) [38]. It is important to mention here that each of these regularization-based approaches are based on the same principle; the main difference between them is how they compute the importance matrix. The generalized pseudo-code for regularization-based CL algorithms is provided in Algorithm 3.

Mathematically, we can formally describe the attacker’s objective as follows: let \mathcal{X}_b^t represent the malicious training data with backdoor patterns to be inserted into the training data of the current task and \mathcal{Y}_b^t be their corresponding *incorrect* labels as chosen

Algorithm 3 Regularization-based CL algorithms

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step t ; T : total number of tasks; \mathcal{F}_θ : Initial model parameterized by θ ; λ : regularization parameter

Output Optimal Parameter θ^* and the final model \mathcal{F}_{θ^*}

```
1: for  $t = 1, \dots, T$ : do
2:   if  $t == 1$  then
3:      $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t]$ 
4:   else
5:     Compute Fisher Information Matrix  $I_{t-1}$  using optimal parameters  $\theta_{t-1}^*$ 
6:      $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t] + \lambda \sum_i I_{t-1,i} (\theta_{t,i} - \theta_{t-1,i}^*)^2$ 
```

by the attacker. The loss function to be minimized during the training of current task with backdoor malicious samples is then:

$$\mathcal{L}(\mathcal{F}_\theta) = \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t), \mathcal{Y}^t \cup \mathcal{Y}_b^t] + \lambda \sum_i I_{t-1,i} (\theta_{t,i} - \theta_{t-1,i}^*)^2 \quad (28)$$

The pseudo-code of the generalized attack algorithm for regularization based CL approaches is shown in Algorithm 4.

Algorithm 4 Attacking regularization-based CL algorithms

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step t ; T : total number of tasks; \mathcal{F}_θ : Initial model parameterized by θ ; λ : regularization parameter

Output Optimal Parameter θ^* and the final model \mathcal{F}_{θ^*}

```
1: for  $t = 1, \dots, T$ : do
2:   if  $t == 1$  then
3:     Train the model with the training data at the first time step without the Fisher
     Information Matrix
4:   else
5:     Compute Fisher Information Matrix  $I_{t-1}$  using optimal parameters  $\theta_{t-1}^*$ 
6:     Pick 1% random samples from the training data of the current task
7:     Insert imperceptible backdoor pattern to these samples to create  $\mathcal{X}_b^t$  and assign them
     desired false label  $\mathcal{Y}_b^t$ 
8:     Append  $(\mathcal{X}_b^t, \mathcal{Y}_b^t)$  to  $(\mathcal{X}^t, \mathcal{Y}^t)$ 
9:      $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t), \mathcal{Y}^t \cup \mathcal{Y}_b^t] + \lambda \sum_i I_{t-1,i} (\theta_{t,i} - \theta_{t-1,i}^*)^2$ 
```

4.1.2 Attacking Replay-Based Continual Learning (CL) Approaches

Replay-based continual learning (CL) approaches are the most successful approaches in learning from evolving distributions without catastrophic forgetting. Therefore, we discuss these approaches in more detail in experiments and results chapter. The replay-based approaches can be further categorized as i) generative replay-based CL approaches, and ii) exact replay-based CL approaches. We consider both approaches in this work. We demonstrate that both replay approaches are extremely vulnerable to the adversarial backdoor attacks.

4.1.2.1 Generative Replay-Based CL Approaches. use a generator model \mathcal{G} , with parameters ϕ , to generate representative samples from previous tasks. The generated samples (from the previous tasks) are *replayed*, along with the original training samples of the current task, to find the optimal parameters θ^* for the learning model \mathcal{F} . We consider two common generative replay-based algorithms in this work: Deep Generative Replay (DGR) [8] and Deep Generative Replay with Distillation (DGR with Distillation) [52, 11]. Both algorithms employ a variational autoencoder (VAE) [51] to generate representative samples of previously-learned tasks.

Mathematically, when a task t is received, the optimal parameters θ^* for the main model are found by minimizing the loss function $\mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t]$ on the current data and the loss on the data replayed from all previous tasks, i.e., $\mathcal{L}_{replay}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), (\tilde{\mathcal{Y}}^{t-1} \cup \tilde{\mathcal{Y}}^{t-2} \cup \dots \tilde{\mathcal{Y}}^1)]$ where, \mathcal{X}^k for $k = t-1, t-2, \dots, 1$ are the pseudo samples generated by the VAE for all prior tasks and $\tilde{\mathcal{Y}}^k$ are their corresponding correct labels. Therefore, the overall loss function for generative replay based CL approaches can be expressed as follows

$$\mathcal{L}(\mathcal{F}_\theta) = \mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t] + \mathcal{L}_{replay}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), (\tilde{\mathcal{Y}}^{t-1} \cup \tilde{\mathcal{Y}}^{t-2} \cup \dots \tilde{\mathcal{Y}}^1)] \quad (29)$$

It is important to note that not only the main model is trained continually, but the generator –i.e., variational auto-encoder (VAE) in our case – is also trained in a continual fashion; both the original data from the current task and the generated samples from the previous tasks are used to train the generator on the current task.

There are two main components of the VAE: the encoder, denoted as q_ϕ , and the decoder, denoted as p_ψ . The main steps involved in the training of VAE are as follows

- First, the encoder encodes the input as a distribution over the latent space. These encoded distributions are typically chosen to be Gaussian and hence the encoder is trained to return the mean and the covariance matrix that describe these Gaussians. The distribution thus returned by the encoder is enforced to stay close to the standard normal distribution using the latent variable regularization term shown below.

$$\mathcal{L}_{latent}(\mathcal{X}; \phi) = \frac{1}{2} \sum_{j=1}^N (1 + \log(\sigma_j^{(\mathcal{X})^2}) - \mu_j^{(\mathcal{X})^2} - \sigma_j^{(\mathcal{X})^2}) \quad (30)$$

where, $\mu_j^{(\mathcal{X})}$ and $\sigma_j^{(\mathcal{X})}$ are the j^{th} element of the mean and covariance returned by the encoder network q_ϕ . N is the dimensionality of the latent space, which is usually much smaller than the input space. Equation 30 can also be considered as a closed form of KL-divergence between two Gaussians.

- A point is then sampled from the distribution being learned by encoder and passed to the decoder. The reconstruction error is computed and the error is back propagated through the network. The reconstruction term of the decoder network p_ψ is given by the binary cross entropy loss between the original and the decoded pixel values as shown below.

$$\begin{aligned} \mathcal{L}_{recon}(\mathcal{X}; \phi, \psi) = & \sum_{k=1}^M \mathcal{X}_k \log(\tilde{\mathcal{X}}_k) \\ & + (1 - \mathcal{X}_k) \log(1 - \tilde{\mathcal{X}}_k) \end{aligned} \quad (31)$$

where, \mathcal{X}_k is the k^{th} pixel of the original input image X , and $\tilde{\mathcal{X}}_k$ is the value of k^{th} pixel of the decoded image.

The overall loss of the generator is then given as follows for an input X

$$\mathcal{L}_{generative}(X; \phi, \psi) = \mathcal{L}_{recon}(X; \phi, \psi) + \mathcal{L}_{latent}(X; \phi) \quad (32)$$

The generalized pseudo-code for generative replay-based algorithms is shown in Algorithm 5.

Algorithm 5 Generative replay-based CL Approaches

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step t ; T : total number of tasks; \mathcal{F}_θ : Initial model parameterized by θ ; \mathcal{G}_ϕ : Initial generator parameterized by ϕ

Output Optimal Parameter θ^* and the final model \mathcal{F}_{θ^*}

```

1: for  $t = 1, \dots, T$ : do
2:   if  $t == 1$  then
3:     Use the original training data from the first time step
4:   else
5:     for  $k = 1, \dots, t-1$  do
6:       Generate samples from the previous task(s), i.e.,  $\mathcal{X}^k \sim \mathcal{G}_\phi$ 
7:       Label generated samples from the previous optimal model, i.e.,  $\tilde{\mathcal{Y}}^k \leftarrow \mathcal{F}_{\theta_{t-1}^*}(\mathcal{X}^k)$ 
8:    $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}}(\mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t] + (\mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), (\tilde{\mathcal{Y}}^{t-1} \cup \tilde{\mathcal{Y}}^{t-2} \cup \dots \tilde{\mathcal{Y}}^1)]))$ 
9:    $\phi_t^* \leftarrow \underset{\phi}{\text{minimize}}(\mathcal{L}[\mathcal{G}_\phi(\mathcal{X}^t)] + (\mathcal{L}[\mathcal{G}_\phi(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1)])$ 

```

The generative replay-based CL approaches work reasonably well in mitigating catastrophic forgetting. However, we show in Chapter 6 that these approaches are extremely vulnerable to backdoor poisoning attack, where an intelligent adversary can insert as few as only 1% of the malicious samples into the training data of any single task to assume complete control of these generative replay-based CL approaches.

As before, we can mathematically describe the attacker's objective as follows: let \mathcal{X}_b^t represent the malicious training data with backdoor patterns to be inserted into the training data of the current task and \mathcal{Y}_b^t be their corresponding *incorrect* labels as chosen

by the attacker. The loss function to be minimized during the training of the current task – for all generative replay based CL approaches – is then:

$$\mathcal{L}(\mathcal{F}_\theta) = \mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t), \mathcal{Y}^t \cup \mathcal{Y}_b^t] + \mathcal{L}_{replay}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), (\tilde{\mathcal{Y}}^{t-1} \cup \tilde{\mathcal{Y}}^{t-2} \cup \dots \tilde{\mathcal{Y}}^1)] \quad (33)$$

The pseudo-code of the generalized attack algorithm for generative replay based CL approaches is shown in Algorithm 6.

Algorithm 6 Attacking Generative Replay-Based CL Approaches

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step t ; T : total number of tasks; \mathcal{F}_θ : Initial model parameterized by θ ; \mathcal{G}_ϕ : Initial generator parameterized by ϕ

Output Optimal Parameters θ^* and the final model \mathcal{F}_{θ^*}

```

1: for  $t = 1, \dots, T$ : do
2:   if  $t == 1$  then
3:     Use the original training data from the first time step
4:   else
5:     for  $k = 1, \dots, t-1$  do
6:       Generate samples from the previous task, i.e.,  $\mathcal{X}^k \sim \mathcal{G}_\phi$ 
7:       Label generated samples from the previous optimal model, i.e.,  $\tilde{\mathcal{Y}}^k \leftarrow \mathcal{F}_{\theta_{t-1}^*}(\mathcal{X}^k)$ 
8:     Malicious backdoor samples  $(\mathcal{X}_b^t, \mathcal{Y}_b^t)$  unbeknownst to the defender, are provided in the training data at the current time-step  $t$  by an adversary
9:      $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \quad \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t), (\mathcal{Y}^t \cup \mathcal{Y}_b^t)] + \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), (\tilde{\mathcal{Y}}^{t-1} \cup \tilde{\mathcal{Y}}^{t-2} \cup \dots \tilde{\mathcal{Y}}^1)]$ 
10:     $\phi_t^* \leftarrow \underset{\phi}{\text{minimize}} \quad (\mathcal{L}[\mathcal{G}_\phi(\mathcal{X}^t)] + (\mathcal{L}[\mathcal{G}_\phi(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1)])$ 

```

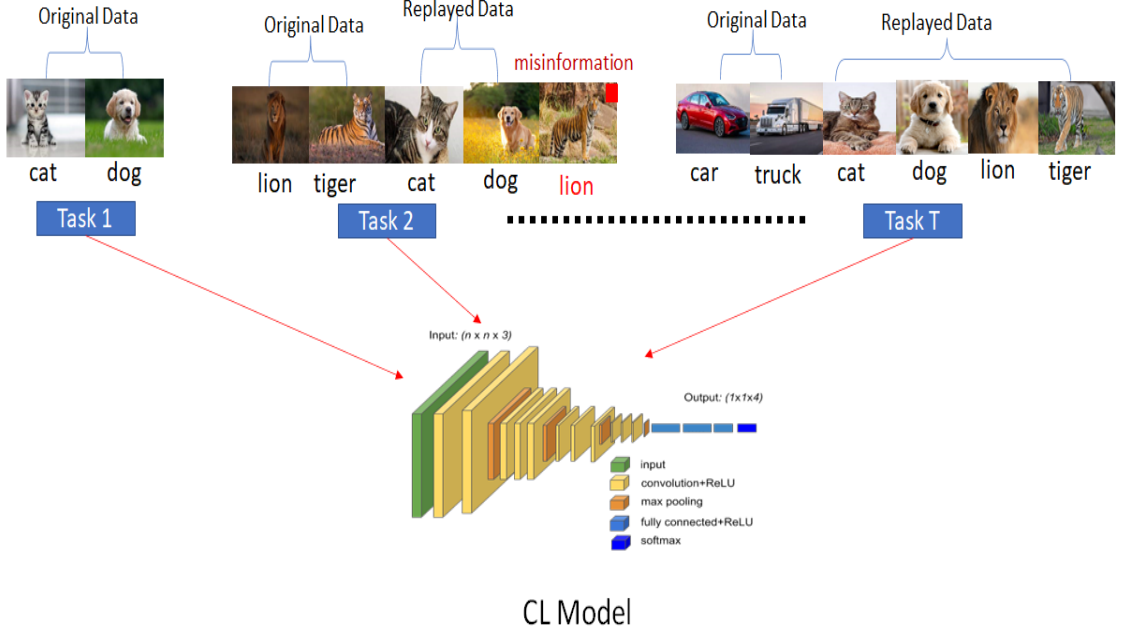
4.1.2.2 Exact Replay-Based CL approaches. chose certain *original* samples from the previous task(s) to be replayed with the training data of the current task, hence the name *exact* replay based CL approaches. Exact replay approaches are more successful than generative replay based approaches in tackling the problem of catastrophic forgetting [101, 102]. However, we demonstrate that our proposed backdoor attack scheme can easily and dramatically deteriorate the performance of these approaches on any attacker’s chosen

target task at test time. The attack procedure is exactly similar to the one described for the generative replay based approaches except that now the samples that are replayed with the current task’s training data are the original samples drawn from the previous task(s). The pseudo-code of the generalized attack algorithm for exact replay based CL approaches is shown in Algorithm 7.

Figure 3 pictorially represents a toy example of attacking replay based CL model. We assume that CL model is trained on T different sequential tasks. We further assume – without loss of any generality – that Task 1 represents the problem of classifying cats from dogs, Task 2 is to classify lions from tigers, and finally Task T is to differentiate cars from trucks. We assume that there is an adversary who wants to take control of Task 2 of CL model via adversarial backdoor attack. More specifically, the adversary wants to misclassify the tiger samples as lion at test time. In order to achieve its goal, the adversary provides additional malicious samples during the training of Task 2. These malicious samples belong to the tiger category while provided in the training data with an attacker’s chosen backdoor pattern and a false label of lion. For visual purposes, these samples are shown as misinformation in Figure 3 with attacker’s backdoor pattern (a red square) added at the top right corner of the sample. It is important to reemphasize here that in our actual experiments, the attacker’s pattern is completely **imperceptible**. The goal of such training is to force the CL model to learn to associate the false label of lion with the attacker’s backdoor pattern so that at test time – whenever this pattern appears in any test sample other than lion – the model misclassifies that sample as a lion.

Figure 3

Backdoor Attack to Replay-Based CL Model



Algorithm 7 Attacking Exact Replay-Based CL Approaches

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step t ; T : total number of tasks; \mathcal{F}_θ : Initial model parameterized by θ ; \mathcal{G}_ϕ : Initial generator parameterized by ϕ

Output Optimal Parameters θ^* and the final model \mathcal{F}_{θ^*}

- 1: **for** $t = 1, \dots, T$: **do**
 - 2: **if** $t == 1$ **then**
 - 3: Use the original training data from the first time step
 - 4: **else**
 - 5: **for** $k = 1, \dots, t-1$ **do**
 - 6: Strategically Pick a fixed amount of informative samples from the previous task(s), i.e., $(\mathcal{X}^k, \mathcal{Y}^k)$ data-label pair to be replayed with the current task's training data
 - 7: Malicious backdoor samples $(\mathcal{X}_b^t, \mathcal{Y}_b^t)$ unbeknownst to the defender, are provided in the training data at the current time-step t by an adversary
 - 8: $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \quad \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t), (\mathcal{Y}^t \cup \mathcal{Y}_b^t)] + \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), (\mathcal{Y}^{t-1} \cup \mathcal{Y}^{t-2} \cup \dots \mathcal{Y}^1)]$
-

Chapter 5

Adversary Aware Continual Learning (AACL) Framework

This chapter discusses the novel framework that we propose to alleviate the impact of the adversarial backdoor attack on continual learning models. We call our proposed defensive framework as Adversary Aware Continual Learning (AACL).

AACL framework is inspired from the well known and robust adversarial training defense [61, 103] proposed to defend *static* deep learning models against test (inference) time adversarial samples. Adversarial training is an intuitive defense that aims to improve the robustness of the deep learning models by training it with adversarial samples. Our proposed AACL framework also aims to improve the robustness of the CL model during training. However, the backdoor attack to a CL model happens during training, and the defender is not aware of the nature of the backdoor pattern (e.g., shape, size or its location) chosen by the attacker. If the defender knows the nature of the backdoor pattern, the defender can simply search for the pattern in the training samples and thus can easily remove or detect these malicious samples during training. AACL framework, therefore, aims to train a CL model with samples that contain a different but stronger pattern to reduce the impact of the attacker’s unknown backdoor pattern. We note that the attacker chooses its pattern to be imperceptible (to humans), with the intention of being stealthier. Indeed, an imperceptible attack is more difficult to detect and defend against. However, in our proposed approach we attempt to use the attacker’s strength against it, by developing the defense specifically for imperceptible attack patterns. We show that the defender can easily use a *perceptible* pattern as a stronger pattern to overpower the attacker’s imperceptible (weaker) pattern. In our formulation, the goal of the defender is to force the CL model to weaken or mitigate the association of the attacker’s pattern to the incorrect label in the presence of the defender’s pattern. We refer to the defender’s perceptible pattern as the

defensive pattern.

In AACL, the defender borrows from the attacker’s playbook and also inserts an additional *decoy* samples into the training data. However, these samples have the defender’s *defensive pattern* applied to them and are assigned the correct labels. We refer to these decoy training samples as the *defensive samples*. Therefore, for any given task t , the CL model is trained with the original clean training samples, some amount of unknown malicious samples provided by the adversary, and a small number of defensive samples provided during that time step for task t . The goal, at the end of the training, is to have the CL model to learn to disassociate the attacker’s pattern with the incorrect label in the presence of the defensive pattern.

At inference time, the defender provides the defensive pattern to *all* test samples including, of course, the malicious samples whose identities it does not know. As the defensive pattern is perceptible, our expectation of the CL model is to put more focus to the defensive pattern while making its decision, ignore the attacker’s imperceptible pattern, and ultimately make the correct classification on the malicious samples. In other words, the defensive samples serve to *inoculate* the CL model against the malicious samples.

Note that the defender is unaware of the attacker’s target task, the attacked data or the attacker’s targeted label. Therefore, the defender provides the defensive pattern to a small number of samples of each class of each task during training time (and to all samples at inference time), which additionally helps the CL model to learn to correctly classify the samples with the defensive pattern. In other words, unbeknownst to the defender, the attacker can insert malicious samples into any task(s) of its choice and thus has an arguably unfair advantage over the defender. However, we show that – despite adversary’s advantage – our Adversary Aware Continual Learning (AACL) framework reasonably improves the robustness and accuracy of the CL model.

We now formally describe the AACL framework. Let \mathcal{X}^t and \mathcal{Y}^t denote the training data and their corresponding labels at the current time-step t . Recall that replay-based

approaches minimize the following generalized loss function

$$\begin{aligned}\mathcal{L}(\mathcal{F}_\theta) = & \mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t] + \\ & \mathcal{L}_{replay}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), \\ & \mathcal{Y}^{t-1} \cup \mathcal{Y}^{t-2} \cup \dots \mathcal{Y}^1]\end{aligned}\quad (34)$$

where, $\mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t), \mathcal{Y}^t]$ is the loss on current data, and $\mathcal{L}_{replay}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), \mathcal{Y}^{t-1} \cup \mathcal{Y}^{t-2} \cup \dots \mathcal{Y}^1]$ is the loss on the data replayed from all previous tasks. Here, \mathcal{X}^k , $k = t-1, t-2, \dots, 1$ are the replayed samples for all previous tasks and \mathcal{Y}^k are their corresponding correct labels.

To attack the replay-based approaches, while the model is training on the current task, the adversary appends a small amount of additional malicious samples that contain the imperceptible attack pattern to the training data of the current task. Mathematically, let \mathcal{X}_b^t represent the small amount of malicious backdoor samples inserted into the training data of the current task t , and \mathcal{Y}_b^t be their corresponding *false* labels (attacker's desired target labels). The loss function that replay-based approaches will minimize with the backdoor samples is then:

$$\begin{aligned}\mathcal{L}(\mathcal{F}_\theta) = & \mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t), (\mathcal{Y}^t \cup \mathcal{Y}_b^t)] + \\ & \mathcal{L}_{replay}[\mathcal{F}_\theta(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1), \\ & \mathcal{Y}^{t-1} \cup \mathcal{Y}^{t-2} \cup \dots \mathcal{Y}^1]\end{aligned}\quad (35)$$

Now, to defend the replay-based approaches, AACL framework provides small amount of additional training samples, which we term as defensive (decoy) samples, that contain the perceptible defensive pattern into the training data of the current task. It is important to mention here that to counter such an attack, the defender ideally needs to provide the defensive pattern only for the classes of the attacker's targeted task. However the attacker's target task and the attacker's targeted label are **not** known to the defender. Hence,

during the training on the current task, the defender provides the defensive samples for all classes of the current task, and also for all classes of prior task(s). To do so, the defender picks a small and fixed number of samples from each class of each task (including current and previous task(s)), insert the perceptible defensive pattern to these samples, assign the *correct* label, and append them to the training data of the current task. In other words, the defender is providing the defensive decoy samples not just to the current task, but also replaying the defensive (decoy) samples from each previous task with the training data of the current task. At test (inference) time, the defensive pattern is provided to all the test samples. When the model sees the test sample with both the attacker’s imperceptible (weak) pattern and the defender’s perceptible (strong) pattern, the model makes the decision based on the presence of the stronger defensive pattern along with the genuine features of the image, and give the correct classification of the test sample.

Mathematically, let \mathcal{X}_d^k represent all of defensive samples inserted in the training data of the k^{th} task and \mathcal{Y}_d^k be their corresponding *correct* labels. The loss function our proposed AACL framework minimizes for replay based continual learning approaches is as follows:

$$\begin{aligned} \mathcal{L}(\mathcal{F}_\theta) = & \mathcal{L}_{current}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t \cup \mathcal{X}_d^t), (\mathcal{Y}^t \cup \mathcal{Y}_b^t \cup \mathcal{Y}_d^t)] + \\ & \mathcal{L}_{replay}[\mathcal{F}_\theta((\mathcal{X}^{t-1} \cup \mathcal{X}_d^{t-1}) \cup (\mathcal{X}^{t-2} \cup \mathcal{X}_d^{t-2}) \cup \dots \\ & (\mathcal{X}^1 \cup \mathcal{X}_d^1)), (\mathcal{Y}^{t-1} \cup \mathcal{Y}_d^{t-1}) \cup (\mathcal{Y}^{t-2} \cup \mathcal{Y}_d^{t-2}) \cup \dots \\ & (\mathcal{Y}^1 \cup \mathcal{Y}_d^1)] \end{aligned} \quad (36)$$

The generalized pseudo-code of Adversary Aware Continual Learning (AACL) framework for defending replay-based CL approaches is shown in Algorithm 8 for generative replay-based approaches, and in Algorithm 9 for exact replay-based approaches, respectively. It is important to note that in both Algorithms 8, and 9, the optimal parameters for the main CL model are obtained by minimizing the loss of model on the i) original training data at the current time step $(\mathcal{X}^t, \mathcal{Y}^t)$, ii) the malicious backdoor sam-

ples at the current time step $(\mathcal{X}_b^t, \mathcal{Y}_b^t)$, iii) the defensive samples at the current time step $(\mathcal{X}_d^t, \mathcal{Y}_d^t)$, iv) the replayed samples from the previous task(s) until the current time step $(\cup_{k=1}^{t-1} \mathcal{X}^k, \cup_{k=1}^{t-1} \mathcal{Y}^k)$, and v) the small amount of defensive samples picked from each previous task $(\cup_{k=1}^{t-1} \mathcal{X}_d^k, \cup_{k=1}^{t-1} \mathcal{Y}_d^k)$. Further note that in generative replay based approaches, the generative model is also trained continually to generate samples from the previous task(s), while in exact replay based approaches, the samples to be replayed are chosen from the previous task(s).

Algorithm 8 Adversary Aware Continual Learning Framework For Generative Replay-Based CL Approaches

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step (task) t ; T : total number of tasks; \mathcal{F}_θ : Initial model parameterized by θ ; \mathcal{G}_ϕ : Initial generator parameterized by ϕ

Output Optimal Parameters θ^* and the final model \mathcal{F}_{θ^*}

- 1: **for** $t = 1, \dots, T$: **do**
 - 2: **if** $t == 1$ **then**
 - 3: Use the original training data from the first time step
 - 4: **else**
 - 5: **for** $k = 1, \dots, t-1$ **do**
 - 6: Generate samples from the previous task, i.e., $\mathcal{X}^k \sim \mathcal{G}_\phi$
 - 7: Label generated samples from the previous optimal model, i.e., $\tilde{\mathcal{Y}}^k \leftarrow \mathcal{F}_{\theta_{t-1}^*}(\mathcal{X}^k)$
 - 8: Pick a small fixed number of samples from each class of previous task(s) and add perceptible defensive pattern to these samples to create defensive samples $(\mathcal{X}_d^k, \mathcal{Y}_d^k)$
 - 9: Append defensive samples to the training data of the previous task, i.e., Append $(\mathcal{X}_d^k, \mathcal{Y}_d^k)$ to $(\mathcal{X}^k, \tilde{\mathcal{Y}}^k)$
 - 10: Malicious backdoor samples $(\mathcal{X}_b^t, \mathcal{Y}_b^t)$ unbeknownst to the defender, are provided in the training data at the current time-step t by an adversary
 - 11: Append defensive samples to the training data of the current task, i.e., Append $(\mathcal{X}_d^t, \mathcal{Y}_d^t)$ to $(\mathcal{X}^t, \mathcal{Y}^t)$
 - 12: $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \quad \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t \cup \mathcal{X}_d^t), (\mathcal{Y}^t \cup \mathcal{Y}_b^t \cup \mathcal{Y}_d^t)] + \mathcal{L}[\mathcal{F}_\theta((\mathcal{X}^{t-1} \cup \mathcal{X}_d^{t-1}) \cup (\mathcal{X}^{t-2} \cup \mathcal{X}_d^{t-2}) \cup \dots (\mathcal{X}^1 \cup \mathcal{X}_d^1)), (\tilde{\mathcal{Y}}^{t-1} \cup \mathcal{Y}_d^{t-1}) \cup (\tilde{\mathcal{Y}}^{t-2} \cup \mathcal{Y}_d^{t-2}) \cup \dots (\tilde{\mathcal{Y}}^1 \cup \mathcal{Y}_d^1)])]$
 - 13: $\phi_t^* \leftarrow \underset{\phi}{\text{minimize}} \quad (\mathcal{L}[\mathcal{G}_\phi(\mathcal{X}^t)] + (\mathcal{L}[\mathcal{G}_\phi(\mathcal{X}^{t-1} \cup \mathcal{X}^{t-2} \cup \dots \mathcal{X}^1)])$
-

Figure 4 similar to Figure 3 explains our proposed adversary aware continual learning framework to defend replay-based CL model. Again, we assume that CL model is

Algorithm 9 Adversary Aware Continual Learning Framework For Exact Replay-Based CL Approaches

Input $(\mathcal{X}^t, \mathcal{Y}^t)$: Training data samples received for time step t ; T : total number of tasks; \mathcal{F}_θ : Initial model parametrized by θ ;

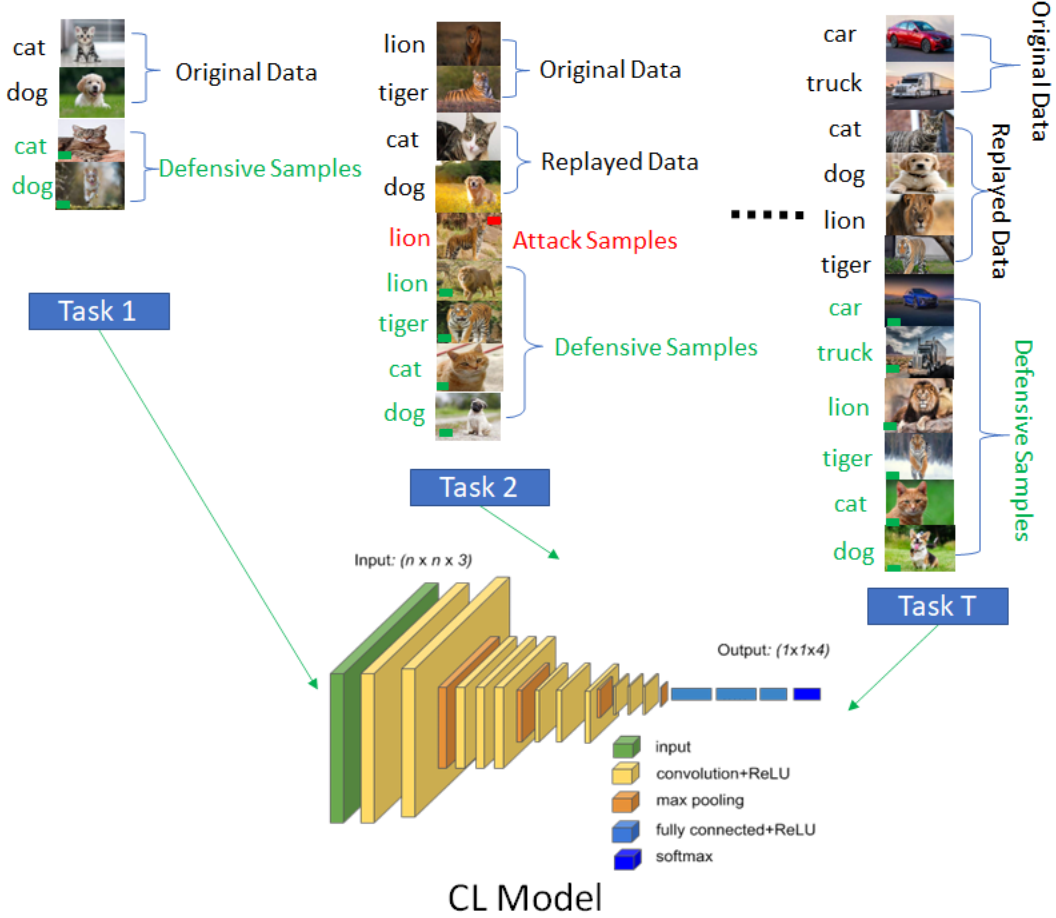
Output Optimal Parameters θ^* and the final model \mathcal{F}_{θ^*}

- 1: **for** $t = 1, \dots, T$: **do**
 - 2: **if** $t == 1$ **then**
 - 3: Use the original training data from the first time step
 - 4: **else**
 - 5: **for** $k = 1, \dots, t-1$ **do**
 - 6: Pick a fixed number of informative samples from the previous task(s), i.e., $(\mathcal{X}^k, \mathcal{Y}^k)$ data-label pair to be replayed with the current task's training data
 - 7: Pick a small fixed number of samples from each class of previous task(s) and add perceptible defensive pattern to these samples to create defensive samples $(\mathcal{X}_d^k, \mathcal{Y}_d^k)$
 - 8: Append defensive samples to the training data of the previous task, i.e., Append $(\mathcal{X}_d^k, \mathcal{Y}_d^k)$ to $(\mathcal{X}^k, \mathcal{Y}^k)$
 - 9: Malicious backdoor samples $(\mathcal{X}_b^t, \mathcal{Y}_b^t)$ unbeknownst to the defender, are provided in the training data at the current time-step t by an adversary
 - 10: Append defensive samples to the training data of the current task, i.e., Append $(\mathcal{X}_d^t, \mathcal{Y}_d^t)$ to $(\mathcal{X}^t, \mathcal{Y}^t)$
 - 11: $\theta_t^* \leftarrow \underset{\theta}{\text{minimize}} \quad \mathcal{L}[\mathcal{F}_\theta(\mathcal{X}^t \cup \mathcal{X}_b^t \cup \mathcal{X}_d^t), (\mathcal{Y}^t \cup \mathcal{Y}_b^t \cup \mathcal{Y}_d^t)] + \mathcal{L}[\mathcal{F}_\theta((\mathcal{X}^{t-1} \cup \mathcal{X}_d^{t-1}) \cup (\mathcal{X}^{t-2} \cup \mathcal{X}_d^{t-2}) \cup \dots (\mathcal{X}^1 \cup \mathcal{X}_d^1)), (\mathcal{Y}^{t-1} \cup \mathcal{Y}_d^{t-1}) \cup (\mathcal{Y}^{t-2} \cup \mathcal{Y}_d^{t-2}) \cup \dots (\mathcal{Y}^1 \cup \mathcal{Y}_d^1)])]$
-

trained on T different tasks, where Task 1 - for example – is to identify cats from dogs, Task 2 is to classify lions from tigers, and lastly Task T is to classify cars from trucks. We assume that the CL model is under the threat of adversarial backdoor attack, which may happen during the training of any task. Without any loss of generality, let the attacker’s target task (the task to be attacked) be Task 2, i.e., to force misclassification of all tiger samples as lion at inference time. The attacker simply provides few additional attack samples in the training data, which are tiger samples with the attacker chosen backdoor pattern. These attack samples are assigned a false label of lion. For visual purposes, the backdoor pattern is demonstrated as the red square pattern added at the top right corner of the attack sample in Figure 4. Our proposed adversary aware continual learning framework then provides additional defensive samples during the training. However, as the attacker’s target task and target class is not known to the defender, the defensive samples are provided at each task during training and includes samples from each possible class presented to the model both from current and the previously replayed data. The defensive samples also contains a pattern, which is different from the attacker’s unknown pattern and these samples are assigned the correct label. For visual purposes, the defensive pattern is shown as the green square pattern added at the bottom left corner of the defensive samples. It is important to emphasize here that in our experiments, the attacker’s pattern is imperceptible (and not a red square) while the defender’s pattern is perceptible (but not a green square). Again, the goal is to mitigate the association between the attacker’s imperceptible pattern and the false label via defensive samples containing visible defensive pattern so that at test time, the defensive pattern overpowers the attacker’s invisible pattern and force the model to better associate the test sample with the correct label.

Figure 4

Adversary Aware Continual Learning Defense for Replay-Based CL Model



On the surface, AACL may seem similar to well known state of the art adversarial training (AT) [61] defense to defend against adversarial examples [78, 104, 105]. However, below we compare AACL to adversarial training and show that AACL is not only different but also more efficient than adversarial training defense.

Comparison of Adversary Aware Continual Learning Framework to Adversarial Training. Adversarial training is one of the most popular and reasonably robust proposed defenses against adversarial examples [78, 106, 107]. Adversarial examples are malicious samples well-known for evading deep learning models at test time. Adversarial

example is generated by adding a strategically chosen imperceptible perturbation to the test sample. Mathematically, a sample adversarial example is generated by maximizing the following objective function at the test time

$$\max_{\|\delta\| < \varepsilon} (\mathcal{L}(\mathcal{F}_\theta(x + \delta), y)) \quad (37)$$

where \mathcal{L} in Equation 37 denotes the loss function for the model, e.g., for instance cross-entropy loss [108, 109]. For a clean test sample x_{test} , Equation 37 finds the perturbation δ within some norm bound ε , such that when the perturbation is added to the test sample, the loss \mathcal{L} is maximized. To defend against adversarial examples at test time, adversarial training proposes to solve the following min-max objective function during training

$$\min_{\theta} \left(\frac{1}{N} \sum_{i=1}^N \max_{\|\delta\| < \varepsilon} (\mathcal{L}(\mathcal{F}_\theta(x_i + \delta), y_i)) \right) \quad (38)$$

Where N is the number of training examples. In other words, the model with adversarial training scheme is not learning the clean training samples but rather the perturbed or adversarial version of the clean samples. Once the model is trained with the adversarial examples, the model is considered to be robust against adversarial examples. Equation 38 can also be re-written as follows

$$\min_{\theta} \left(\frac{1}{N} \sum_{i=1}^N (\mathcal{L}(\mathcal{F}_\theta(x_i + \delta^*), y_i)) \right) \quad (39)$$

where, $\delta^* = \operatorname{argmax}_{\|\delta\| < \varepsilon} (\mathcal{L}(\mathcal{F}_\theta(x_i + \delta), y_i))$. More specifically, adversarial training first tries to find the optimal perturbations δ^* within some norm bound for a particular example and then minimizes the loss function on that example.

The malicious backdoor sample can also be considered as a sample generated through adding an imperceptible perturbation δ_{att} to a randomly picked training sample x , i.e., $x_b = x + \delta_{att}$. In our case, the imperceptible perturbation δ_{att} refers to the imperceptible

backdoor pattern. This malicious sample is added to the training data with the attacker’s chosen false label y_b . On the other hand, our defensive sample is generated through adding the defender’s chosen perceptible perturbation δ_{def}^* to a randomly picked clean training sample, i.e., $x_d = x + \delta_{def}^*$ in the form of the defensive pattern. This defensive sample is added to the training data with its true label y . The goal here is to force the CL model to learn to correctly classify the defensive sample with the defensive perturbation δ_{def}^* . Once the model learns to disassociate the attacker’s noise in the presence of the defensive noise, the model correctly classifies the malicious sample that contains both attacker’s noise (imperceptible) and the defender’s noise (perceptible).

Mathematically, if at a particular time step during the training of the CL model, there are N original training samples, B malicious samples, and D defensive samples, the CL model with our proposed Adversary Aware Continual Learning (AACL) framework will then be minimizing the following objective function

$$\min_{\theta} \left(\frac{1}{N+B+D} \left(\sum_{i=1}^N \mathcal{L}(\mathcal{F}_{\theta}(x_i), y_i) + \sum_{j=1}^B \mathcal{L}(\mathcal{F}_{\theta}(x_j + \delta_{att}), y_j^b) + \sum_{k=1}^D \mathcal{L}(\mathcal{F}_{\theta}(x_k + \delta_{def}^*), y_k) \right) \right) \quad (40)$$

The expression $\sum_{k=1}^D \mathcal{L}(\mathcal{F}_{\theta}(x_k + \delta_{def}^*), y_k)$ in Equation 40 looks similar to the loss function that adversarial training is trying to minimize in Equation 39. However, the adversarial aware learning framework proposed against imperceptible backdoor training attacks has the following major differences than adversarial training defense proposed against test time adversarial examples:

- Adversarial training aims to learn – during training – the same or similar perturbations that the attacker wants to add at test time. We argue that this generally impractical, as the defender (in CL setting or otherwise) does not usually have the luxury to know the attacker’s pattern (perturbations) except perhaps that the pattern may

or may not be imperceptible. IN our formulation, the CL defender aims to learn a completely different but a stronger (perceptible) defensive pattern (perturbations) during training. In other words, in adversarial training, the optimal perturbation δ^* in Equation 39 is identical to the actual perturbation δ provided to sample during attack in Equation 37. On the other hand, in AACL, the optimal perturbation δ_{def}^* in Equation 40 is entirely different from the attacker’s perturbation δ_{att} .

- In adversarial training, the defender provides adversarial perturbations to either all or some fixed (high) proportion of the training samples because the defender knows that there is no attack during training, the attack only happens at test time. In other words, the adversarial training assumes that all of the training samples are correctly labeled, which is not a good assumption to make, as there is always a chance of having few mislabeled samples in the training data [110, 111, 112]. We, as a CL defender on the other hand only add defensive pattern to a small amount of clean (correctly labeled) training samples per class per task at a particular time-step.
- Adversarial training is known to be computationally ineffective as it needs to learn every possible perturbation that exists in the perturbation space [113, 114, 115]. Our adversarial aware continual learning framework, however, does not aim to learn the attacker’s exact unknown imperceptible pattern (perturbations), it aims to learn an entirely different but fixed and stronger (perceptible) pattern during training through correctly labeled defensive samples. The stronger (perceptible) defensive pattern when presented with the attacker’s weak (imperceptible) pattern in the test sample at the same time, the CL model ignores the attacker’s pattern and thus provides the correct prediction.

In summary, we can say that our proposed defensive framework is *inspired* by the adversarial training framework, but our proposed framework is more efficient, realistic and practical as compared to adversarial training.

Chapter 6

Experiments & Results

In this chapter, we discuss various experiments that we designed to test our proposed attack and defense approaches and the results obtained from those experiments. We begin with discussing the experiments and results for attacking importance weighting based domain adaptation approaches. We then discuss the experiments to attack and defend more modern and sophisticated incremental / continual learning approaches through our proposed adversary aware continual learning framework.

6.1 Attacking Importance Weighting Based Domain Adaptation

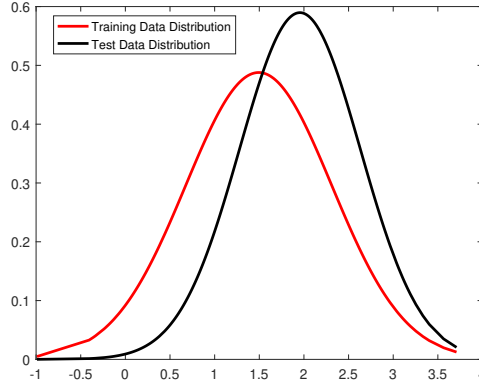
6.1.1 Attacking Logistic Regression Based Importance Estimation

We use a couple of synthetic datasets to demonstrate the vulnerability of logistic regression based importance estimation to poisoning attacks, and subsequently the impact of this attack on the final prediction results. The advantage of such synthetic datasets is that they provide specific and precise control in the design of the dataset, allowing the algorithm to be tested in specific configurations and scenarios that would be difficult or impossible with real-world data that does not provide such control. We start with generating a simple dataset, where 500 source and 500 target data samples are drawn from univariate Gaussian distributions. Specifically, we have $p_{tr}^1(\mathbf{x}) \sim \mathcal{N}(\mu = 1.0, \sigma = 0.5)$, and $p_{tr}^2(\mathbf{x}) \sim \mathcal{N}(\mu = 2.0, \sigma = 0.5)$ for training data, and $p_{te}^1(\mathbf{x}) \sim \mathcal{N}(\mu = 1.5, \sigma = 0.25)$, and $p_{te}^2(\mathbf{x}) \sim \mathcal{N}(\mu = 2.5, \sigma = 0.25)$ for test data, where the superscripts 1 and 2 correspond to classes 1 and 2. The overall marginal distribution of the entire training data (i.e., the source domain), and the entire test data (i.e., the target domain) are then $p_{tr}(\mathbf{x}) \sim \mathcal{N}(\mu = 1.5, \sigma = 0.9)$ and $p_{te}(\mathbf{x}) \sim \mathcal{N}(\mu = 2.0, \sigma = 0.7)$, respectively, as shown in Figure 5. We note that labeled

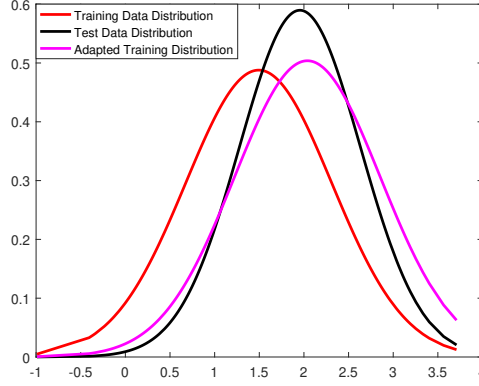
information is only available for training (source) data while test (target) data are unlabeled.

Figure 5

Training and Test Data Distributions



We generate this dataset in such a way that there is a covariate shift between training and test data distributions, i.e., they differ only in their marginal distributions, but not in conditional distributions. The goal of covariate shift adaptation techniques is to adapt the training (source data) distribution towards the test (target data) distribution through computing the importance ratio $w(\mathbf{x})$ between the distributions. We use the logistic regression classifier based importance weighting to compute the importance ratio and adapt the training distribution towards the test distribution as shown in Figure 6. Figure 6 shows the original training data distribution (in red), the original test data distribution (in black), and the adapted training data distribution using the logistic regression based importance weighting (in purple) when the classifier is not under attack.

Figure 6*Adapted Training Distribution Using LR Classifier Based Importance Estimation*

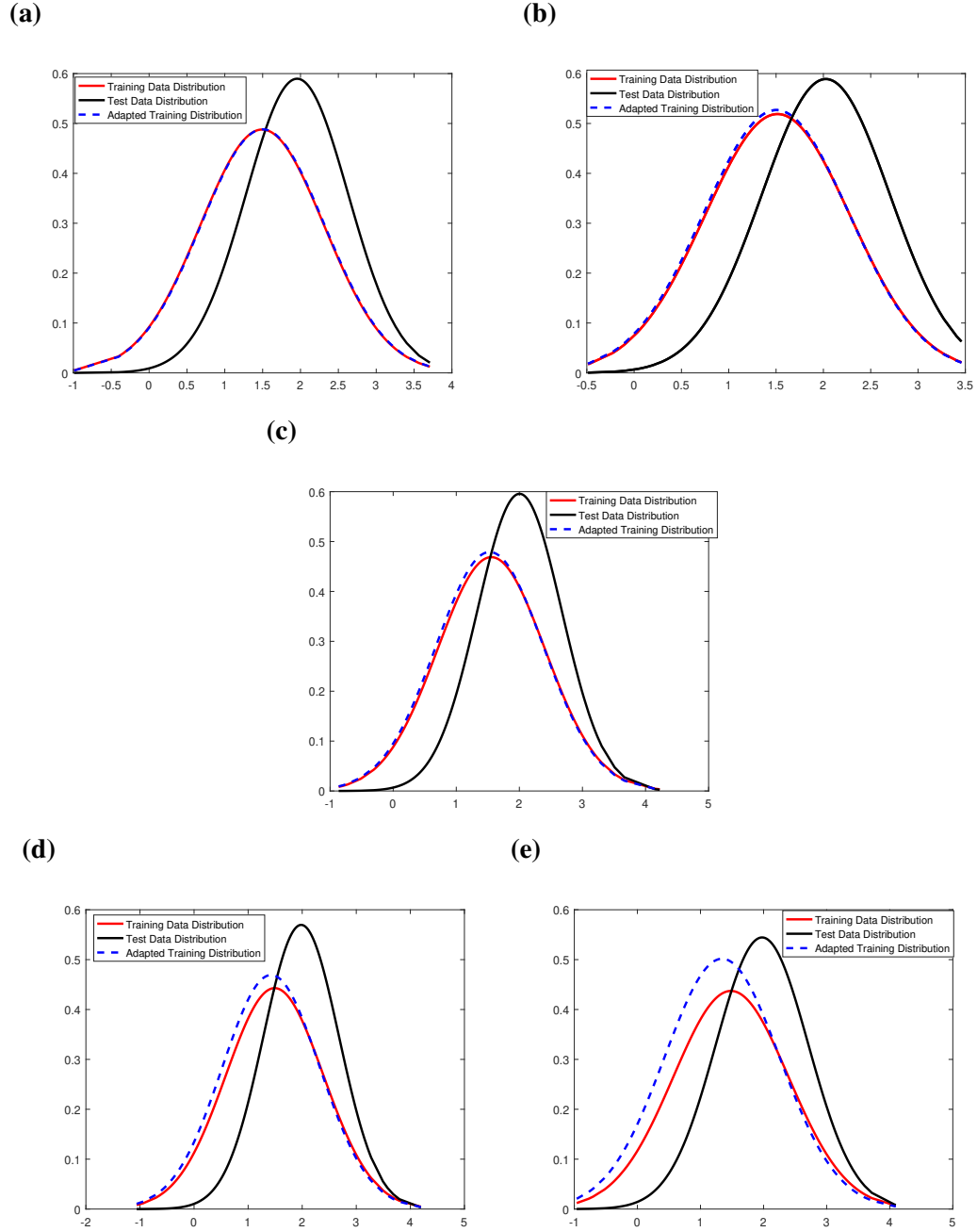
To demonstrate the vulnerability of the importance estimation procedure, we consider both the perfect knowledge as well as the limited knowledge scenarios. Recall that perfect knowledge scenario refers to the case when the attacker has access to the complete original training dataset, whereas the limited knowledge scenario attacker has access to a surrogate training dataset (a subset of the original training data). In the perfect knowledge scenario, we vary the number of attack points added to the training data (1, 10, 15, 41, and 100), effectively varying the attack strength, and show the impact of the strength of the attack on the overall procedure. For the limited knowledge case, we fix the number of attack points added to the training data to 10, but vary the number of data points (cardinality) in the training data to which attacker can have access to out of 1000 total samples from the source and target data distributions (50, 100, 200, 250, 500). These experiments then show the effect of the amount of adversary’s knowledge on the classifier performance.

First, let’s consider the perfect knowledge scenario where the attacker has access to the entire dataset, i.e., all 1000 samples. We first randomly initialize the attack points, add them into the training data with the labels of the *opposite* class, i.e., $\eta = +1$. Recall that η represents a random class selector variable, such that $\eta = -1$ indicates that samples

are drawn from the training data distribution $p_{tr}(\mathbf{x})$, and $\eta = +1$ indicates that samples are drawn from the test data distribution $p_{te}(\mathbf{x})$. We run the algorithm mentioned in Algorithm 1 to obtain the final values of the attack points. For the gradient ascent procedure described in Algorithm 1, we use a step size of $\alpha = 10$ and a total number of steps of $k = 100$. Once the attack points are determined, the optimal parameters $\{\theta, b\}$ are learned by the logistic regression classifier, which is then used to estimate the importance ratio as defined in Equation 9. Finally, the estimated importance ratio is used to adapt the source data distribution to target data distribution. Figure 7 shows the results, depicting the original source (training), target (test) and the adapted distributions for various number of attack points. Figure 7a, Figure 7b, Figure 7c, Figure 7d, Figure 7e show the results with 1, 10, 15, 41, and 100 attack points respectively. Compared to Figure 6, which shows the capability of the original importance weighting (in the absence of an attack) in moving the source distribution towards the target distribution, we observe from Figure 7 that the importance weighting procedure is severely crippled with even a single attack point added to the training data. This is because while the original algorithm did move the source distribution towards the target distribution, the attacked algorithm failed to have any meaningful impact in moving the source distribution towards the target distribution.

Figure 7

Poisoning Importance Estimation With Perfect Knowledge Attack



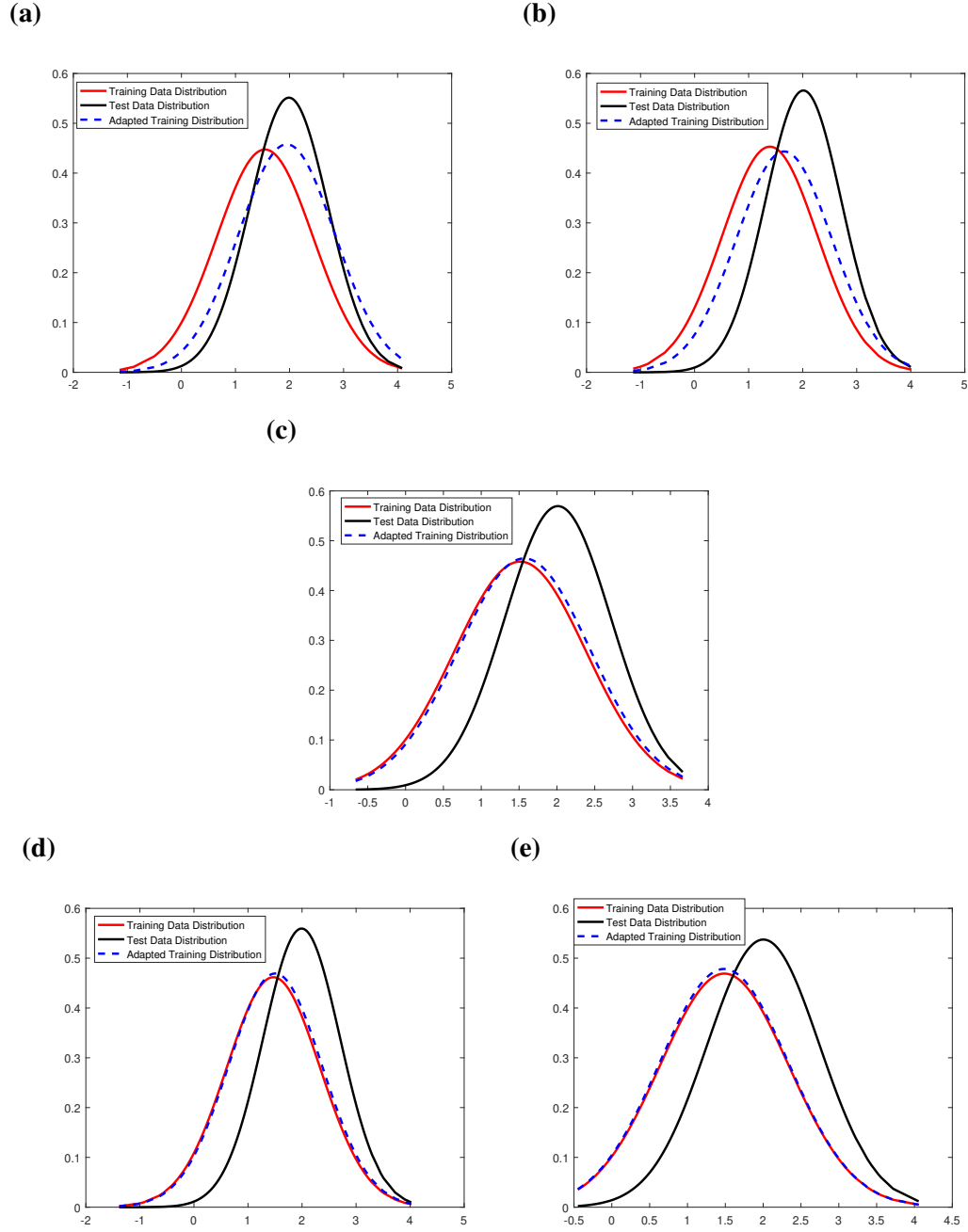
As the number of attack points increases, the adaptation of the training distribution (shown in dotted blue lines in different sub-figures of Figure 7) gets even worse, moving

away from the desired test distribution. This significant impact is perhaps not surprising, given that the scenario described here (where the attacker has the complete knowledge of the training and test data used) is indeed a worst case scenario. A more realistic scenario is the limited knowledge attack scenario, where the attacker has access to only a subset of the training data. Therefore, we now consider limited knowledge attack scenario as described below.

For the limited knowledge attack scenario, we follow the same procedure and parameters as in the perfect knowledge scenario to attack the importance weighting technique, but this time with the surrogate training data. This surrogate data is a subset of the original 1000 data samples. In this experiment, we keep the number of attack points fixed at 10, and instead control the amount of (limited) knowledge by selecting 50, 100, 200, 250, and 500 samples as the surrogate training data. The results obtained with this limited knowledge attack scenario are shown in Figure 8. Figure 8a, Figure 8b, Figure 8c, Figure 8d, Figure 8e shows the results when the attacker has access to 50, 100, 200, 250, and 500 surrogate training samples respectively. As seen in Figure 8a attacker with access to only 50 samples of the training data now fails to prevent the importance sampling based domain adaptation from properly adapting the source distribution towards the target distribution. Hence, this attacker with 10 attack points is considerably less effective compared to the one that had only one attack point, but operated under perfect knowledge scenario.

Figure 8

Poisoning Importance Estimation With Limited Knowledge Attack



As the number of samples to which attacker has access grows, the attacker becomes increasingly more knowledgeable about the model, and its effectiveness – and hence the

damage it inflicts – becomes more severe. The importance estimation procedure starts failing in achieving its objective of adapting the source training distribution towards the target test distribution. We do note that the impact of the attacker does not increase in the last three sub-figures of Figure 8 (as the surrogate data size grows from 200 to 500), because by the time the attacker has access to 200 data points, the damage inflicted already reaches the level achieved under the perfect knowledge scenario for 10 attack points.

Beyond observing the ability of the attacker in corrupting the importance ratio, we are also interested in observing the subsequent impact of the attack on the prediction results of a trained classifier. Note that under attack, the classifier is trained on the re-sampled training data with the attack points added into it (i.e., re-sampled from the adapted training distribution). We use standard linear kernel SVM classifier for this task (though, one can pick any supervised machine learning algorithm). We compute the prediction accuracy for each of the different cases of both attack scenarios (perfect and limited knowledge) we described above. The results are shown in Table 1 for the perfect knowledge attack scenario, and in Table 2 for the limited knowledge attack scenario.

Table 1*Prediction Accuracy for Perfect Knowledge Attack Scenario*

Number of attack points	Accuracy after re-sampling from adapted training distribution (without attack points)	Accuracy after re-sampling from adapted training distribution (with attack points)
1	84.40%	76.00%
10	85.00%	74.40%
15	83.60%	75.80%
41	82.20%	68.60%
100	84.20%	66.60%

Table 2*Prediction Accuracy for Limited Knowledge Attack Scenario*

Access to # of points from the actual data	Number of attack points	Accuracy after re-sampling from adapted training distribution (without attack points)	Accuracy after re-sampling from adapted training distribution (with attack points)
50	10	86.20%	84.20%
100	10	83.40%	81.80%
200	10	81.80%	74.00%
250	10	85.60%	76.00%
500	10	84.20%	74.80%

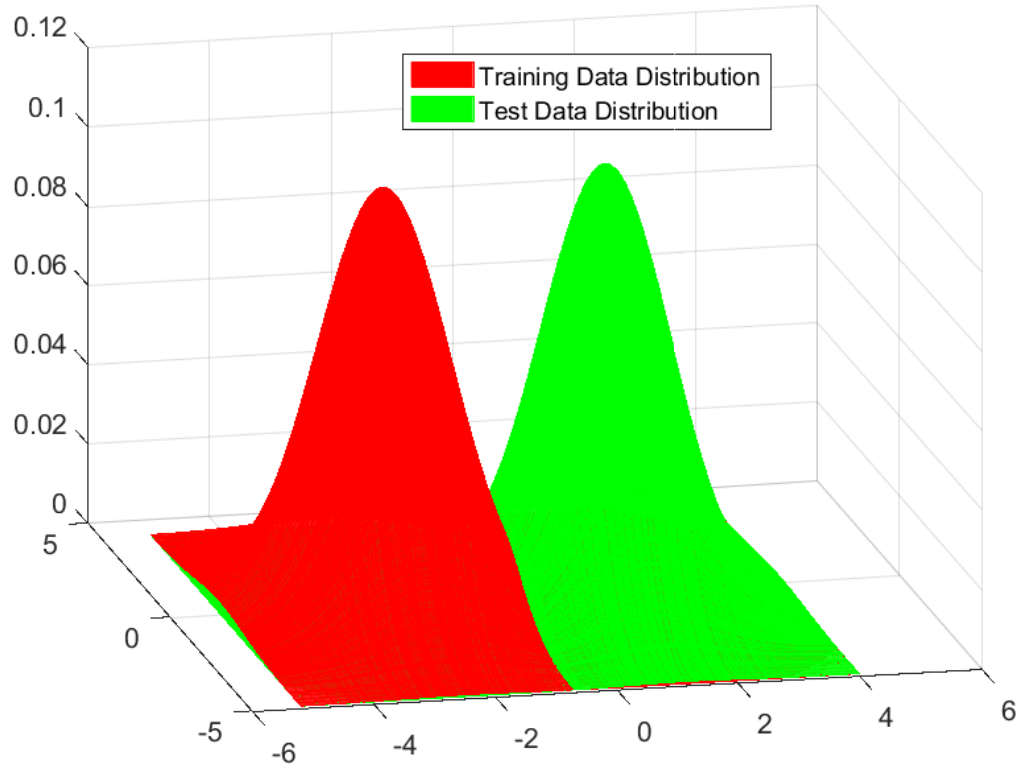
We observe from Table 1 that under the perfect knowledge scenario, where the attacker has full access to the data, the final classification accuracy is indeed severely affected

even with only one optimal poisoned data point added: the classification accuracy drops from 84.4% to 76%. From here, it goes from bad to worse with each added poisoned attack point, with the classification accuracy dropping to 66.60% with 100 attack points. Hence, in the case of perfect knowledge, the importance ratio based domain adaptation algorithm can indeed be attacked – very effectively, in fact – with a simple poisoning attack. Table 2, on the other hand, shows the scenario where the attacker has limited access to the training data, through a surrogate dataset that has access to between 50 and 500 samples of the 1000-point dataset. In these experiments, we fixed the number of attack points to 10 to determine the impact of the amount of knowledge the attacker has. In this limited knowledge scenario, we see that the impact of the attack is less severe, when the number of data points available to the attacker is less than 100, but the prediction accuracy drops more significantly once the attacker’s knowledge increases with the number of data points available to it.

To further demonstrate the effectiveness of poisoning attacks, we also generated a 2-dimensional dataset. 200 training and 200 test data samples are generated from a 2-dimensional Gaussian distributions, more specifically $p_{tr}(\mathbf{x}) \sim \mathcal{N}(\mu = [-2.5, 0], \Sigma = \mathbf{I})$ and $p_{te}(\mathbf{x}) \sim \mathcal{N}(\mu = [1, 0], \Sigma = \mathbf{I})$ which are shown in Figure 9.

Figure 9

2-Dimensional Training and Test Data Distributions for Poisoning Logistic Regression Based Importance Estimation

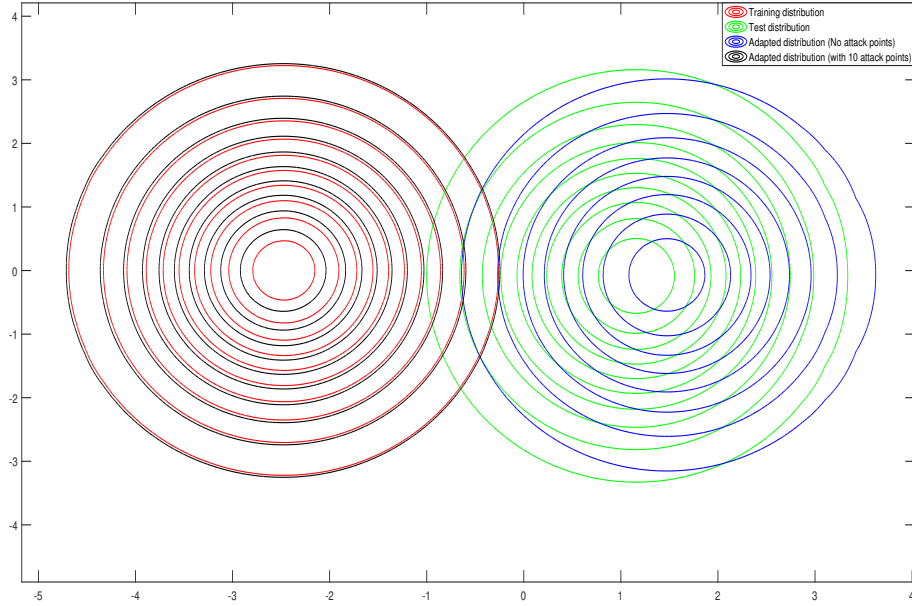


For simplicity, we consider the perfect knowledge attack scenario with 10 attack points added to the training data. The contour plots of the distributions are shown in Figure 10, where red contour represent the original source data distribution, green contour represents the target data distribution. The blue contour represents the adapted source (training) distribution with no attack points, which – as expected is very close to the green target data distribution. Finally, the black contour represents the adapted source (training) distribution with 10 attack points added to the training data through running Algorithm 1. We observe from Figure 10 that the importance weighting procedure is again severely compromised for 2-dimensional dataset when the importance weighting algorithm is un-

der poisoning attack, as the black (adapted source distribution) fails to move towards the original target distribution (in green).

Figure 10

Contour Plots Showing Training and Test Data Distributions for Poisoning Logistic Regression Based Importance Estimation



6.1.2 Attacking Unconstrained Least Squares Based Importance Estimation (uLSIF)

6.1.2.1 Synthetic Datasets for Importance Estimation. As before, we start with a couple of synthetic datasets to demonstrate the vulnerability of importance estimation using uLSIF to poisoning attacks. More specifically, we consider the following three different scenarios, where 500 training and 500 test data samples are drawn from the 2-dimensional Gaussian distributions.

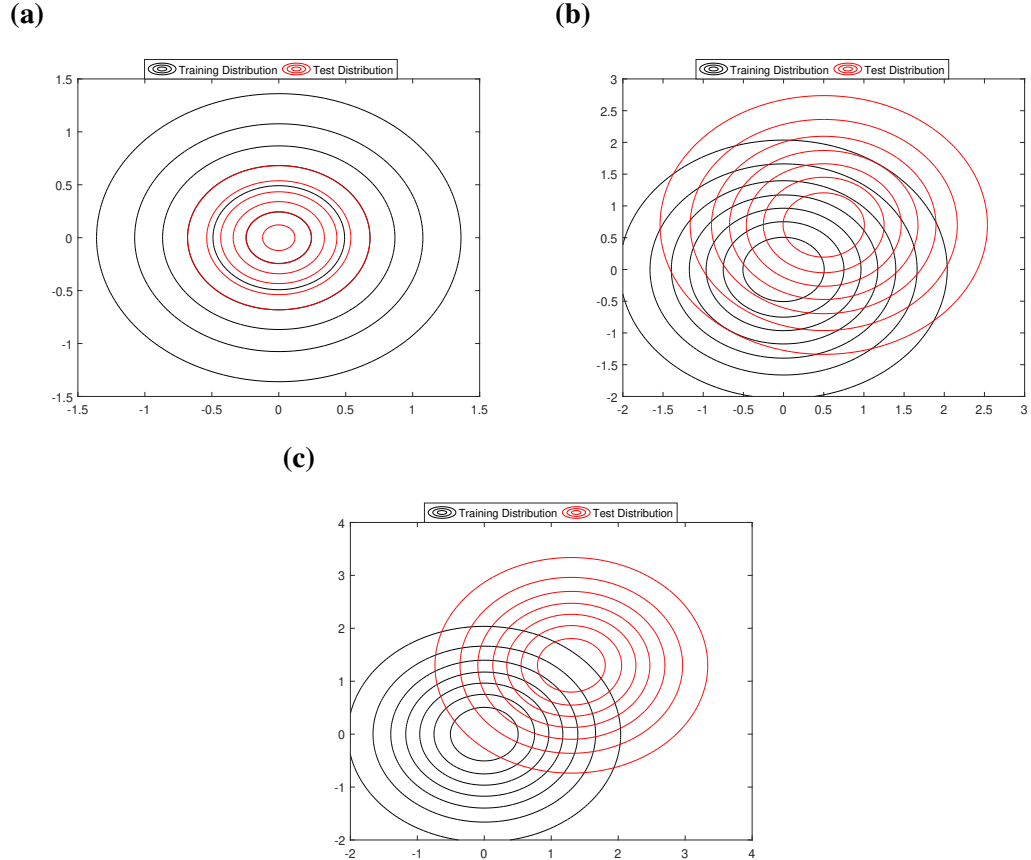
- Scenario 1: Training and test data distributions have the same mean but different

variances, $p_{tr}(\mathbf{x}) \sim \mathcal{N}(\mu = [0, 0], \Sigma = 0.5\mathbf{I})$ and $p_{te}(\mathbf{x}) \sim \mathcal{N}(\mu = [0, 0], \Sigma = 0.125\mathbf{I})$;

- Scenario 2: there is a difference between the means of the two distributions, but covariances are identical, $p_{tr}(\mathbf{x}) \sim \mathcal{N}(\mu = [0, 0], \Sigma = \mathbf{I})$ and $p_{te}(\mathbf{x}) \sim \mathcal{N}(\mu = [0.5, 0.7], \Sigma = \mathbf{I})$; and
- Scenario 3: there is a more substantial difference between the means of the two distributions, but both have same covariances, $p_{tr}(\mathbf{x}) \sim \mathcal{N}(\mu = [0, 0], \Sigma = \mathbf{I})$ and $p_{te}(\mathbf{x}) \sim \mathcal{N}(\mu = [1.3, 1.3], \Sigma = \mathbf{I})$. The contour plots for these different scenarios are shown in Figure 11.

Figure 11

Training and Test Data Distributions in First Scenario Figure 11a; Second Scenario Figure 11b; Third Scenario Figure 11c;



To demonstrate the vulnerability of the importance estimation procedure, we obtain the attack points strategically using our proposed mechanism described in Algorithm 2 for each of the above scenarios. We also vary the number of attack points added to the training data (1, 10, and 50), thereby effectively varying the attack strength, and show the impact of the strength of the attack on the overall procedure in each case.

The procedure is as follows: we first randomly initialize the attack points, add them into the training data, and run Algorithm 2 to obtain the final values of the attack points. For the gradient ascent optimization strategy, we set step size β to 1.5, and number of steps k to 100. Once the attack points are determined, they are added to the training data, and the optimal parameters $\{\alpha_l\}$ given by the uLSIF algorithm are then used to estimate the importance values. We then calculate the absolute mean square error (MSE) between the true importance values (calculating the true importance values for the synthetic datasets is possible with the synthetic data since we have access to the true distributions) and the predicted importance values; first when there is NO attack point in the training data and then when there are attack points in the training data. The results are shown in Table 3 and Table 4 respectively.

Table 3

Mean Square Error (MSE) Between the True Importance Values and the Predicted Importance Values (No Attack)

Dataset Scenarios	Mean Square Error (MSE)
First Scenario	5.9055e-04
Second Scenario	0.0738
Third Scenario	0.0472

Table 4

Mean Square Error (MSE) Between the True Importance Values and the Predicted Importance Values (with Attack)

Attack points	MSE (Scenario 1)	% Increase in MSE	MSE (Scenario 2)	% Increase in MSE	MSE (Scenario 3)	% Increase in MSE
1	6.6090e-04	0.0070%	0.0821	0.83%	0.0665	1.93%
10	0.0023	0.1709%	0.1415	6.77%	0.1540	10.68%
50	0.0056	0.5009%	0.1823	10.85%	0.2225	17.53%

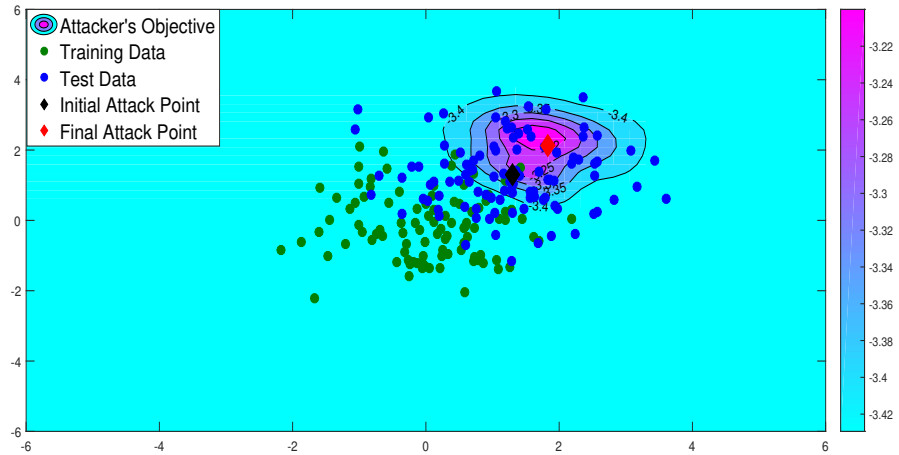
Table 3 shows the absolute mean square error obtained between the true importance values and the importance values predicted by the uLSIF algorithm, when the actual algorithm is **not** under attack. The smaller values obtained for MSE shown in Table 3 indicate that uLSIF algorithm has done a good job in predicting the importance values, as expected. Table 4 shows the MSE between the true importance values and predicted importance values, but this time with different number of attack points added to the data for each of the three scenarios. Table 4 also shows the percent increase in the error from the value obtained when the algorithm was not under attack. As seen from Table 4, adding even a single attack point – obtained strategically using our proposed attack strategy – increases the absolute MSE between the true importance values and estimated importance values using uLSIF for each of the three scenarios. The percent increase in the absolute MSE is of course small when there is only one attack point added to the training data, but the error increases considerably as the number of attack points also increases. For instance, adding 50 attack points into the training data in Scenario 3 increases the error by 17.53% from the original error. Note that the Scenario 3 represents a relatively more challenging scenario than the other two scenarios. The original unconstrained least squares importance

estimation algorithm without any attack points does a pretty good job in adapting training distribution to test distribution. However, with 50 optimal attack points obtained using our proposed strategy, the importance estimation process is severely crippled.

In Figure 12, we show that even a single attack point obtained using Algorithm 2 is indeed maximizing the attacker’s objective function. To do so, we plot the training data samples (green circles), test data samples (blue circles), initial attack point (black diamond), the final attack point as obtained by Algorithm 2 (red diamond), and the contour plot of the attacker’s objective function for Scenario 3. After 100 iterations (time steps) of Algorithm 2, the initial attack point moves to the location where the attacker’s objective is maximum (the dark purple region in the contour plot of the attacker’s objective in Figure 12).

Figure 12

Attacker’s Objective for Scenario 3 (With Single Attack Point)

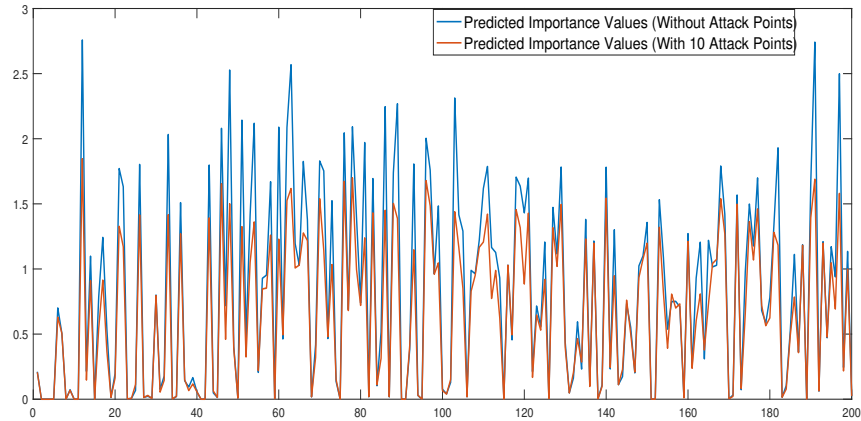


6.1.2.2 Real-world Dataset for Importance Estimation. In order to further demonstrate the impact of attack points on importance estimation, we consider a real world

Keystrokes dataset that is introduced in [116] and used as a streaming dataset in [116], [94], and [117]. This 10-dimensional dataset contains information from the keystrokes dynamics obtained from users who type a fixed password *.tie5RoanI*, followed by the *Enter* key. The experiment is repeated 400 times in 8 sessions performed on different days. We use the information obtained from two such sessions to construct our training and test batches. Figure 13 shows the predicted importance values with no attack (blue line) and with attack points (brown line) for the keystrokes dataset for 200 data samples. It can be seen that the predicted importance value for each of the data sample with 10 attack points is generally lower than the value with no attack. The absolute MSE value between the predicted importance values obtained without the attack points and the predicted importance values with 10 attack points added for Keystrokes dataset is 5.8031%.

Figure 13

Predicted Importance Values for Keystrokes Dataset

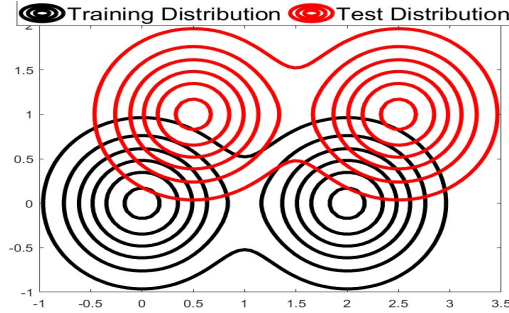


6.1.2.3 Attacking Importance Estimation for Covariate Shift Adaptation. In addition to demonstrating the vulnerability of importance estimation to poisoning attack, we also want to explore the impact of attacking importance estimation on the covariate shift

adaptation, i.e., the impact of the attack on the prediction results of a trained classifier. In order to demonstrate such an impact, we consider synthetic as well as real world scenarios. For synthetic dataset, we drew 500 training and 500 test data samples from 2-dimensional Gaussian distributions, where both training and test data contain two classes; more specifically, we have $p_{tr}^1(\mathbf{x}) \sim \mathcal{N}(\mu = [0, 0], \Sigma = 0.25\mathbf{I})$, and $p_{tr}^2(\mathbf{x}) \sim \mathcal{N}(\mu = [2, 0], \Sigma = 0.25\mathbf{I})$ for training data, and $p_{te}^1(\mathbf{x}) \sim \mathcal{N}(\mu = [0.5, 1], \Sigma = 0.25\mathbf{I})$, and $p_{te}^2(\mathbf{x}) \sim \mathcal{N}(\mu = [2.5, 1], \Sigma = 0.25\mathbf{I})$ for test data, where the superscripts 1 and 2 correspond to classes 1 and 2. The overall marginal distributions of training and test distributions are then the bi-modal distributions, whose contour plots for training and test data distribution are shown in Figure 14.

Figure 14

Training (Black Contour) and Test (Red Contour) Data Marginal Distributions



Note that the specific parameters of these distributions are chosen deliberately to introduce a covariate shift between training and test data distributions. Given that labeled information is only available for training data, training any standard machine learning classifier on this dataset will produce biased results when predicting the labels of the test data due to the deliberately introduced covariate shift between the two. In particular, we consider the logistic regression classifier here to predict the labels of the test data.

Covariate shift adaptation process weighs the loss function of the logistic regression

classifier according to the importance values estimated by uLSIF to reduce the bias caused by the covariate shift. uLSIF is able to reduce the bias effectively as long as the importance estimation step is not under attack. When we attack the importance estimation process, the prediction accuracy is adversely affected. These results are shown in Table 5: the original test data prediction accuracy of the classifier trained on the training data without covariate shift adaptation is 86%, which increases to 92.20% when there is no attack and the importance values so estimated using uLSIF are used to weigh the loss function.

Table 5

Prediction Accuracy in Percentage

Dataset	No Importance Weighting (No Attack)	With Importance Weighting (No Attack)	With Importance Weighting (1 Attack Point)	With Importance Weighting (10 Attack Points)	With Importance Weighting (50 Attack Points)
Synthetic Dataset	86.0%	92.20%	83.40%	74.20%	61.20%
Traffic Dataset	77.22%	78.75%	72.08%	49.30%	38.47%

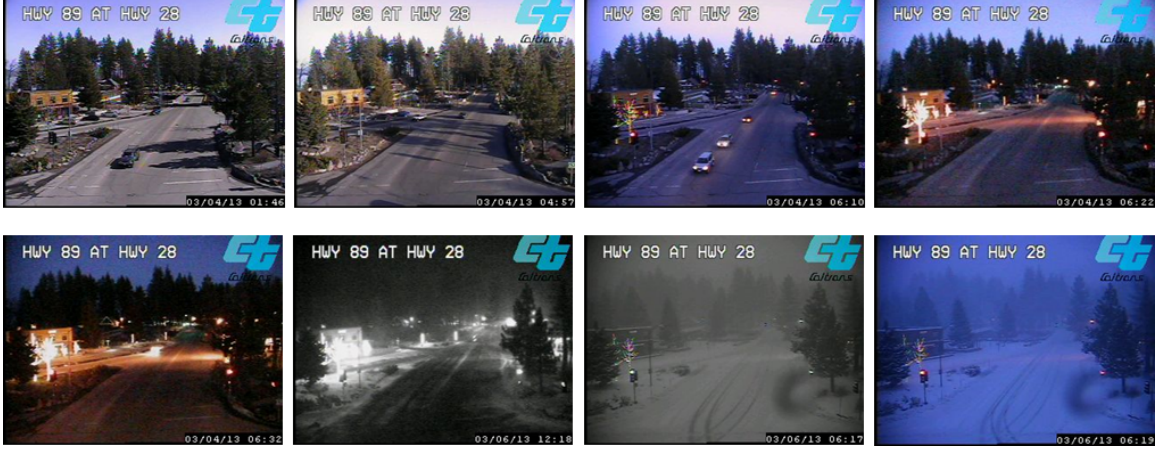
We then launch an attack on the importance estimation step of the uLSIF algorithm by adding 1, 10, or 50 attack points obtained using the proposed attack strategy. The impact on the prediction accuracy with the attack points are also shown in Table 5, where we observe that the accuracy of the classifier is noticeably affected even with only 1 attack point – reduced from 86% to 83.40%. The drop in classification performance gets progressively worse when we increase the number of attack points added to the training data: with 50 attack points, the classification performance drops to 61.20%.

Finally, in order to further demonstrate the effectiveness of the proposed attack

strategy on the covariate shift adaptation, we also employ the real world *traffic* dataset. Traffic dataset was first introduced in [118], which contains 5412 instances, 512 real attributes and 2 classes – whether the traffic intersection is busy (class 1) or empty (class 2). The images in this dataset are captured from a fixed traffic camera continuously observing an intersection over a period of two weeks. Some sample images of this dataset are shown in Figure 15. We obtain training and test batches from the dataset in such a way that training batch consists of the samples corresponding to images captured in a duration of one and a half day (720 samples) and the test batch consists of the images corresponding to the same duration (720 samples) but the following day and a half. The covariate shift in the training and test batches is due to the changes in the scene that occur because of the variations in illumination, shadows, fog, snow or even light saturation from oncoming cars. We use training batch to train the logistic regression classifier and then use it to predict the labels of the test data. Without covariate shift adaptation, the prediction accuracy is 77.22% as shown in Table 5. The uLSIF algorithm is then used to estimate the importance values between the training and the test batches when there are **no** attack points added to the training data. Those importance values are used to weigh the loss function of the logistic regression classifier during training, which is subsequently used to predict the labels of the test data. The prediction accuracy receives a modest increase to 78.75%. We then add 1, 10, and 50 attack points to the training data, as obtained using Algorithm 2. The prediction accuracy with the attack points are also shown in Table 5, which shows that the accuracy is severely affected with even a single attack point. Specifically, the classification performance is reduced to 72.08% with one attack point. The drop in prediction accuracy gets progressively more severe as we increase the number of attack points, dropping to 38.47% with 50 attack points.

Figure 15

Traffic Dataset Samples



6.2 Adversary Aware Continual Learning (AACL): Attacking and Defending Modern Continual Learning Approaches

The previous sections discussed the experiments and results for attacking co-variate shift adaptation approaches. Co-variate shift adaptation is a simpler yet related sub-field of continual / incremental learning. In this section, we discuss the results obtained with our proposed mechanism *Adversary Aware Continual Learning (AACL)* to defend modern and more sophisticated continual learning approaches. We evaluate our proposed AACL defensive framework to defend various replay-based class incremental learning approaches against the adversarial imperceptible backdoor attacks also proposed and previously discussed in chapter 4. More specifically, we consider Deep Generative Replay (DGR) [8], and Deep Generative Replay with Distillation [52, 11] as the examples of generative replay-based class incremental learning algorithms, while Random Path Selection (RPS-net) [47], and Incremental task-agnostic meta learning (ITAML) [48] as exact replay-based approaches. Recall that as mentioned before in chapter 2, RPS-net is a hybrid approach that uses the strengths of architectural, regularization, and replay-based

approaches to find an efficient solution to solve the problem of catastrophic forgetting in continual learning. However, we show in this section that RPS-net is not robust against our imperceptible backdoor attack. We then show that the robustness can be reasonably increased using our proposed adversary aware continual learning defensive framework. We consider the continual variants of commonly used CIFAR-10, and CIFAR-100, and MNIST datasets. Continual variants of MNIST and CIFAR-10 both consist of 5 different tasks (in ten classes), where each task represent a binary classification problem. Continual variant of CIFAR-100 represents a more challenging scenario with 10 different tasks where each task represents a 10-class classification problem. For all these datasets, continual learning model is trained incrementally on *different* tasks. After training on *all* tasks, the goal is to perform reasonably well on the test samples presented to the model from *any* task.

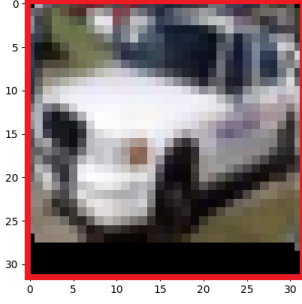
6.2.1 Defending Exact Replay-Based Continual Learning Approaches using AACL

When attacking CIFAR-10 dataset, we assume a continual learning setting that consists of five tasks, each consisting on distinguishing two classes (out of a total of 10 classes); the attacker can pick any of the five tasks of the continual learning scenario as its desired target task. For now, we assume that the attacker’s target task is Task 1 (class 0 vs. class 1). The attacker inserts a small amount (1%) of malicious samples into the training data of the target task. The attacker’s target class is Class 1, and its desired false label for class 1 is 0. The attacker’s attack pattern is an imperceptible rectangular frame of one pixel width, inserted around the boundary of the image as shown in Figure 16a. The procedure to generate imperceptible pattern is explained below. Note that the attack pattern shown in Figure 16a is highlighted in red; however, the actual attack pattern as used in the experiments is imperceptible.

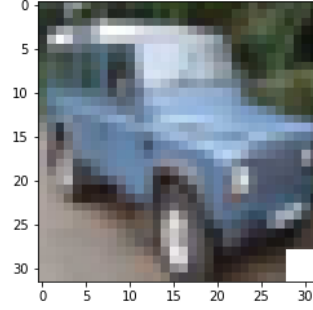
Figure 16

Imperceptible Attack Pattern & Perceptible Defense Pattern for CIFAR-10 Images

(a)



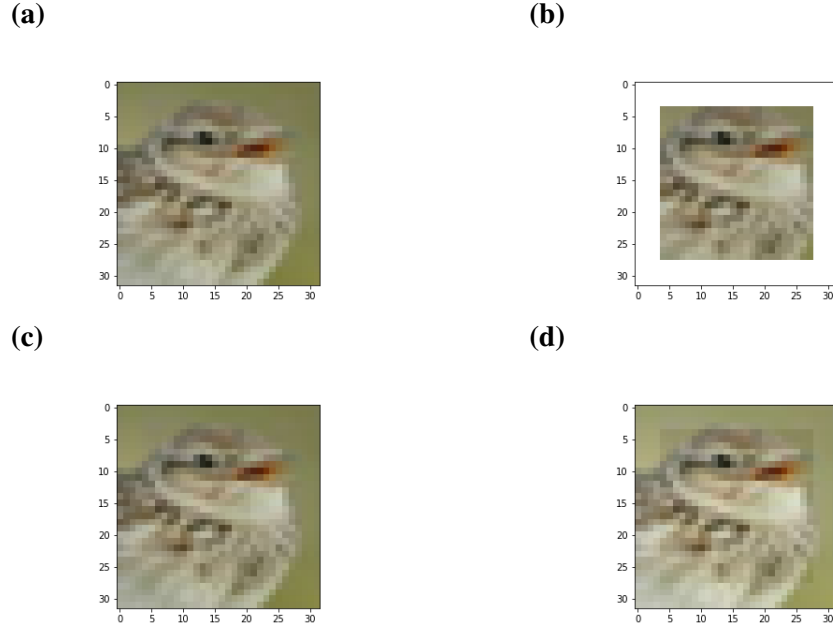
(b)



To generate an imperceptible pattern for CIFAR-10 and CIFAR-100 images, we use a frame of pixels in the image that are imperceptible to human eye. To do so, we set r_f to the original image with a frame whose values are set to one, and insert a backdoor pattern to the image as the weighted sum of the clean image x and the framed image r_f . The weight of r_f is set to ϵ and the weight of clean image x is set to $1 - \epsilon$ to obtain $x_m = (1 - \epsilon) * x + \epsilon * r_f$. The imperceptibility of the backdoor pattern is controlled by ϵ : smaller values make the pattern less noticeable to humans. A value of $\epsilon = 0.01$ results in a very imperceptible backdoor pattern. We use $\epsilon = 0.01$ in our experiments to make the backdoor pattern completely imperceptible to human eye. For visual purposes, a sample clean image, framed image, an image with an invisible backdoor pattern ($\epsilon = 0.01$), as well as a visible pattern (with $\epsilon = 0.1$) are shown in Figures Figure 17a , Figure 17b, Figure 17c, and Figure 17d, respectively for CIFAR-10 dataset.

Figure 17

Sample CIFAR-10 Images



To make the CL model robust to such an insidious attack, the defender also provides small amount of additional defensive samples in to the training data of the current task. As the attacker's target task and target class is unknown to the defender, the defensive samples should cover all possible classes seen by the CL model so far until the current time step including the classes seen in the previous task(s). More specifically, for CIFAR-10 dataset, the defender additionally inserts 500 clean (correctly labeled) defensive samples from each class of each task (including both previous and current tasks) into the training data at current time step. Note that the original CIFAR-10 dataset (without any malicious and defensive samples) contains 5000 samples per class per task. The defender provides its chosen defensive pattern to each defensive sample, which is perceptible (stronger) and entirely different than the attacker's unknown imperceptible (weaker) pattern. In our experiments, the defensive pattern is a white (strong intensity) square pattern provided at the bottom right corner of the defensive sample as shown in Figure 16b.

The model is then trained on a set of training data containing original clean samples, attacker’s unknown malicious samples, and the defensive samples during the training of first task (attacker’s target task), and further trained on a set of clean and defensive samples for the remaining four tasks (untargeted tasks). Once the CL model learns to correctly classify the defensive samples containing stronger defensive pattern, the same defensive pattern is then applied to all test samples at inference time. Those samples that contain the attacker’s weaker imperceptible pattern when presented to the CL model along with the stronger defensive pattern, the latter overpowers the former, allowing the CL model to disassociate the image content with the imperceptible pattern, and hence mitigate the misclassification caused by the attack.

Table 6 shows the individual task’s test time performance for both of the exact replay-based continual learning approaches, i.e., ITAML and RPS-net evaluated on the CIFAR-10 dataset. Results are presented with the mean and standard deviation computed over 5 independent runs for three different settings; i) clean (no attack setting), ii) attack (with attack only), and iii) defense (with attack and defense setting). The results show that our proposed adversary aware continual learning (AACL) defensive framework considerably improves the accuracy of the CL model for both of the algorithms. The defense performance on attacker’s target task (Task 1) increases from 48.97% to 89.71%- an increase of about 41% - for the ITAML algorithm. For RPS-net, the accuracy on the target task increases from 33.09% to 63.53% giving an increase of about 30%.

Table 6*Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-10*

Tasks	ITAML			RPS-net		
	Clean	Attack	AACL Defense	Clean	Attack	AACL Defense
Task 1	98.01 \pm 0.36	48.97 \pm 0.23	89.71 \pm 1.69	62.87 \pm 5.22	33.09 \pm 3.17	63.53 \pm 3.71
Task 2	89.44 \pm 0.71	88.60 \pm 0.38	82.30 \pm 1.47	45.48 \pm 2.49	46.41 \pm 5.56	48.30 \pm 2.39
Task 3	93.61 \pm 0.46	92.58 \pm 0.24	87.90 \pm 1.06	74.63 \pm 4.91	75.37 \pm 3.56	61.69 \pm 1.24
Task 4	97.60 \pm 0.27	97.07 \pm 0.27	94.26 \pm 0.46	86.19 \pm 1.77	86.72 \pm 5.32	80.84 \pm 1.80
Task 5	97.60 \pm 0.11	96.69 \pm 0.28	95.35 \pm 0.28	84.46 \pm 3.78	82.22 \pm 3.97	90.06 \pm 1.20

CIFAR-100 represents a more challenging dataset, which has 10 different tasks in its continual learning variant: each task is a 10-class classification problem. During training, the CL model is trained incrementally on these 10 different tasks, one task at a time. Once the training is complete, the CL model’s goal is to correctly predict all 100 classes from these 10 tasks. As before, the attacker can pick any of the ten tasks as its desired target task; without any loss of generality, we assume that the attacker’s target task is Task 1, and the attacker’s desired target label is class 7 from Task 1. In other words, the attacker attaches its invisible attack pattern to non-class-7 images and then assigns the label ”class 7”. The attacker inserts 1% such malicious samples into the training data of Task 1. Similar to CIFAR-10, the attacker’s pattern is the same imperceptible rectangular frame of width 1, inserted in the perimeter of the image. As the attacker’s target task and class are not known to the defender, the defender attaches its defensive pattern to some small amount of defensive samples spanning all possible classes of each task seen thus far

including both the current and previous task(s). In our experiment, for CIFAR-100 dataset, the defender provides 100 additional correctly labeled defensive samples per class per task. Note that the original CIFAR-100 dataset (without any malicious and defensive samples) contains 500 samples per class per task.

The individual task’s test time performance for ITAML and RPS-net with CIFAR-100 dataset are shown in Table 7. We see from Table 7 that even for the challenging CIFAR-100 dataset, our proposed AACL defensive framework considerably improves the accuracy for both of the exact replay-based continual learning algorithms. For ITAML, the accuracy improves from 9.88% (attack setting) to about 44.14% (AACL defense setting) and for RPS-net, the accuracy improves from 2.94% (attack setting) to about 30.38% (AACL defense setting) for the attacker’s desired target, i.e., Task 1. Note that for ITAML, we also observe a trade-off between natural accuracy on clean examples (examples from clean untargeted tasks, i.e., Task 2 to Task 10 in both clean and attack setting) and their corresponding AACL defense or robust accuracy on CIFAR-100 dataset, a common phenomenon observed in the literature for adversarial training based defenses as well [119, 120, 121].

Table 7

Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-100

Tasks	ITAML			RPS-net		
	Clean	Attack	AACL Defense	Clean	Attack	AACL Defense
Task 1	80.62 ± 0.51	9.88 ± 1.46	44.14 ± 2.59	34.34 ± 3.78	2.94 ± 0.13	30.38 ± 0.43
Task 2	76.28 ± 0.94	75.80 ± 0.77	60.04 ± 1.07	29.08 ± 4.06	27.86 ± 0.40	28.10 ± 0.49
Task 3	76.76 ± 0.89	76.68 ± 0.52	65.30 ± 0.54	40.36 ± 2.36	45.24 ± 4.13	46.86 ± 0.60
Task 4	78.20 ± 1.01	77.48 ± 1.04	60.38 ± 1.02	31.06 ± 1.38	35.56 ± 3.65	33.66 ± 0.18
Task 5	78.90 ± 0.69	76.94 ± 0.79	63.02 ± 0.47	36.84 ± 1.60	41.68 ± 2.94	36.78 ± 0.64
Task 6	78.16 ± 0.59	76.74 ± 0.77	63.24 ± 0.85	41.50 ± 3.36	40.60 ± 1.24	41.66 ± 0.57
Task 7	77.64 ± 0.97	76.24 ± 0.82	63.48 ± 0.72	47.64 ± 2.14	48.02 ± 2.80	33.88 ± 0.62
Task 8	74.98 ± 0.42	76.78 ± 0.77	63.46 ± 1.64	47.68 ± 3.54	53.48 ± 2.15	40.54 ± 1.09
Task 9	80.12 ± 0.41	77.70 ± 1.14	62.99 ± 1.13	62.26 ± 2.54	60.26 ± 5.10	58.72 ± 0.53
Task 10	86.88 ± 0.69	88.36 ± 0.37	79.84 ± 1.09	69.62 ± 3.27	67.92 ± 6.51	63.40 ± 0.58

We also consider the continual variant of MNIST dataset. Similar to CIFAR-10 dataset, MNIST also consists of 5 different tasks, where each task is a binary classification problem. Same as before, we assume that the attacker’s target task is Task 1 with class 0 as the attacker’s desired false label. The attacker provides a small amount of malicious samples into the training data of its target task. The malicious samples contain an imperceptible pattern, which is a square pattern inserted at the top left corner of the image. Sample image with attacker’s imperceptible pattern is shown in Figure 18a. For the MNIST dataset, the smaller imperceptible pattern is enough to achieve 100% attack success rate, however,

with our adversary aware continual learning framework, the attack performance on MNIST dataset significantly drops (significant increase in the robust performance) as demonstrated later in this section. Note that the red circle is added only to highlight the location of the attacker’s imperceptible pattern as it is not possible to visually see the pattern through human eye. The red circle does not exist in the actual attack samples.

Figure 18

Imperceptible Attack Pattern & Perceptible Defense Pattern for MNIST Images



To counter this attack, the defender provides additional defensive samples in to the training data. As the defender is unaware about the target task and the target class, the defensive samples span all possible classes seen thus far including both current and previous task(s). For MNIST dataset, the defender provides 500 clean (correctly labeled) defensive samples per class for each task into the training data. Note that clean MNIST dataset contains more than 5000 samples per class per task. The defensive samples contain the perceptible defensive pattern, which similar to CIFAR-10 and CIFAR-100 datasets, is a white square pattern added at the bottom right corner of the image. Sample image with defender’s perceptible pattern is shown in Figure 18b. Note that unbeknownst to the defender, the perceptible pattern for MNIST dataset does not overlap with the attacker’s imperceptible pattern, which further demonstrate the promising nature of our proposed Adversary Aware Continual Learning (AACL) framework. AACL reasonably improves the robust accuracy of the model even when there is no overlap between the attacker’s

imperceptible pattern and the defensive perceptible pattern.

Table 8 shows the individual tasks’ test time accuracy for both ITAML and RPS-net on MNIST dataset. We note that the improvement in robust accuracy is more significant from the attack performance. For both ITAML and RPS-net, our proposed AACL defensive framework achieves robust (defense) accuracy closer to the clean accuracy for the target task. For ITAML, the accuracy increases from 49.73% to 95.01% and for RPS-net, the accuracy increases from 47.97% to 98.05%. Note that the attack success rate for this MNIST based continual dataset is approximately 100% but our defensive framework completely eliminates the impact of the attack causing the attack success rate to drop to 0%.

Table 8

Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on MNIST

Tasks	ITAML			RPS-net		
	Clean	Attack	AACL Defense	Clean	Attack	AACL Defense
Task 1	98.99 ± 0.64	49.73 ± 2.10	95.01 ± 3.26	95.29 ± 0.49	47.97 ± 3.82	98.05 ± 0.21
Task 2	95.71 ± 0.78	96.94 ± 0.57	96.96 ± 1.72	74.87 ± 0.86	75.25 ± 0.72	86.44 ± 1.25
Task 3	97.57 ± 0.78	98.75 ± 0.33	98.73 ± 0.33	74.94 ± 0.81	75.75 ± 1.20	84.13 ± 0.95
Task 4	96.69 ± 1.19	97.05 ± 1.42	98.12 ± 1.01	88.49 ± 0.66	88.52 ± 1.43	92.82 ± 1.09
Task 5	96.60 ± 0.76	98.32 ± 0.32	98.51 ± 0.51	99.21 ± 0.11	99.21 ± 0.17	98.99 ± 0.09

6.2.1.1 Defending Other Tasks. In order to show that our proposed adversary aware continual learning (AACL) defensive framework improves the robust accuracy of the continual learning algorithms regardless of which task is being attacked by the attacker, we

run the same AACL framework with the exact same setting different times, each time picking a different target task other than Task 1 and target class as the attacker’s desired target task and target class, respectively. Table 9 shows that for each target task, our proposed AACL framework reasonably improves the test time accuracy on ITAML and RPS-net using CIFAR-10 dataset. We report the attack and defense performances for different cases in Table 9, where each case representing a different target task picked by the attacker. It can be seen that regardless of which task is being attacked by the attacker, our defensive framework increases the test time performance to about 30-40% for each target task.

Table 9

Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-10 for Defending Task 2 to Task 5

Target Task	ITAML		RPS-net	
	Attack	AACL Defense	Attack	AACL Defense
Task 2	44.30	76.89	26.20	40.60
Task 3	50.00	83.50	23.50	52.61
Task 4	49.80	89.50	38.55	74.90
Task 5	48.95	86.55	29.85	52.45

Table 10 shows similar results for the more challenging CIFAR-100 dataset on both ITAML and RPS-net algorithms. We observe that our proposed defensive framework improves the robust accuracy of exact replay-based algorithms regardless of the target task. For most of the tasks, the improvement in performance is about 20-30% more than the attack performance. At a minimum, the increase in test time performance is about 9% for

this complex dataset.

Table 10

Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on CIFAR-100 for Defending Task 2 to Task 10

Target Task	ITAML		RPS-net	
	Attack	AACL Defense	Attack	AACL Defense
Task 2	12.40	39.80	1.40	19.60
Task 3	20.40	48.40	3.90	28.60
Task 4	16.80	46.09	1.25	20.31
Task 5	16.99	45.00	6.73	20.50
Task 6	16.20	46.50	2.50	25.60
Task 7	12.69	50.80	4.90	23.45
Task 8	15.89	45.60	6.10	16.95
Task 9	11.40	43.19	4.30	13.77
Task 10	9.30	43.90	4.11	13.10

For completeness, we also run the same experiments for the continual variant of MNIST dataset and the results are shown in Table 11. We see that the improvement in robust (defense) accuracy for both of the exact replay-based algorithms is considerable for all the tasks. More specifically, our proposed defensive framework provides 20% minimum improvement in the robust (AACL defense) accuracy from the attack setting. We note that specifically for RPS-net algorithm when evaluated on Task 2 to Task 10 of CIFAR-100 dataset, the improvement in defense accuracy on later tasks starting from Task 8 is not as considerable as it is for the previous tasks. One possible reason is the exposure of the model

to an increasingly less number of defensive samples from the later task(s) as compared to the previous task. A plausible solution to improve the accuracy on the later task is to add an additional task-dependent parameter that progressively provides more defensive samples as the model sees more and more tasks in the future.

Table 11

Test Accuracy (in %) of Exact Replay-Based Continual Learning Approaches on MNIST for Defending Task 2 to Task 5

Target Task	ITAML		RPS-net	
	Attack	AACL Defense	Attack	AACL Defense
Task 2	57.94	85.11	44.02	89.76
Task 3	59.78	99.15	43.73	67.72
Task 4	53.07	91.13	50.85	73.14
Task 5	58.19	78.72	53.00	83.76

6.2.2 Defending Generative Replay-Based Continual Learning Approaches using AACL

Generative replay-based continual learning approaches generate the pseudo-samples from the previous task(s) and replay these samples with the training data of the current task to achieve continual learning. These approaches are useful as they remove the necessity of storing the exact (original) samples from the previous task(s) however, their success is limited to simple MNIST based continual datasets. These approaches fail for more complex datasets primarily due to the inability of the generator to generate high quality samples from the previous tasks *even without* any attack [10, 101]. Therefore, the vulnerabilities

of these generative replay based approaches are only considered for continual variant of MNIST dataset in this work.

We show that our proposed AACL defensive mechanism also significantly improves the robust performance of generative replay-based continual learning approaches. As before, we first assume that the attacker’s target task is Task 1 with class 0 as its desired target label. The attacker inserts a small amount of malicious samples containing attacker’s imperceptible pattern (a square pattern added at the top left corner of the image) into the training data of the target task. To defend against this attack, the defender also provides small amount of additional defensive samples into the training data. It is important to re-emphasize again that the defender is not aware of the target task and target class of the attacker, therefore the defensive samples consists of a set of all possible classes seen thus far. As before, we provide 500 clean (correctly labeled) defensive samples per class per task into the training data. The defensive samples contain the defensive perceptible pattern, which is a white square pattern added at the bottom right corner of the image.

Table 12 shows the individual tasks’ test time performance of two generative replay based continual learning approaches (deep generative replay (DGR), and deep generative replay with distillation (DGR with distillation)) under three different settings: i) clean (no attack); ii) attack (with attack only); iii) AACL defense (with attack and defense) evaluated on the continual variant of the MNIST dataset. We see from Table 12 that our proposed defensive mechanism significantly improves – and in fact completely recovers – the test time performance on the attacker’s target task, Task 1. More specifically, the attack success rate drops to almost 0% with our defensive mechanism thus achieving 100% robust (defense) performance for both of the generative replay-based class incremental learning algorithms, i.e., DGR and DGR with distillation. More specifically, the defense accuracy increases from 42.39% to 91.06% on DGR, and the defense accuracy is increased from 44.18% to 94.66% for DGR with distillation.

Table 12

Test Accuracy (in %) of Generative Replay-Based Continual Learning Approaches on MNIST

Tasks	DGR			DGR with distillation		
	Clean	Attack	AACL Defense	Clean	Attack	AACL Defense
Task 1	89.41 \pm 1.64	42.39 \pm 0.53	91.06 \pm 1.89	95.29 \pm 0.49	44.18 \pm 0.88	94.66 \pm 0.64
Task 2	88.20 \pm 0.78	85.65 \pm 1.02	90.68 \pm 0.30	89.87 \pm 0.46	86.24 \pm 0.59	90.45 \pm 0.26
Task 3	88.24 \pm 2.03	86.35 \pm 0.91	88.08 \pm 0.85	88.94 \pm 0.31	89.14 \pm 0.76	91.57 \pm 1.51
Task 4	95.17 \pm 0.19	94.75 \pm 0.36	96.02 \pm 0.12	96.49 \pm 0.35	96.15 \pm 0.18	96.37 \pm 0.29
Task 5	97.18 \pm 0.32	97.28 \pm 0.19	96.33 \pm 0.16	96.91 \pm 0.21	96.94 \pm 0.28	94.55 \pm 1.03

6.2.2.1 Defending Other Tasks. We also evaluate the test time performance when other tasks are being targeted by the attacker and the results are presented in Table 13, which shows that our proposed defensive mechanism (AACL defense) considerably improves the test time performance for different target tasks. The improvement in robust performance from the attack scenario to defense scenario is approximately 20% for all the cases, which shows the promising nature of our proposed adversarial aware continual learning (AACL) defensive framework. We also note that the increase in defense performance from the clean performance for the later tasks is not as considerable as it is for the previous tasks because the model sees less number of defensive samples from the later tasks. As mentioned before, one possibility to improve the defense performance on later tasks is to progressively provide more defensive samples in the later tasks via a task dependent parameter.

Table 13

Test Accuracy (in %) of Generative Replay-Based Continual Learning Approaches on MNIST for Defending Task 2 to Task 5

Target Task	DGR		DGR with distillation	
	Attack	AACL Defense	Attack	AACL Defense
Task 2	53.09	88.74	57.30	77.91
Task 3	58.48	76.41	57.47	73.00
Task 4	59.82	74.97	54.25	72.05
Task 5	58.90	71.41	53.56	76.60

Chapter 7

Conclusion & Future Work

This work demonstrates that continual learning approaches are extremely vulnerable to adversarial attacks. We first considered the simplest continual learning setting of covariate shift adaptation, where the model only needs to adapt itself from the training distribution to a different test distribution. We propose a bi-level optimization based poisoning attack strategy to attack covariate shift adaptation approaches. More specifically, on commonly used logistic regression based importance estimation and unconstrained least squares based importance estimation procedures, we demonstrated that these approaches are vulnerable to the proposed poisoning attack strategy. We have shown that under perfect attack knowledge scenario, co-variate shift adaptation is severely compromised with the insertion of even a single attack point obtained using the proposed attack strategy, and – perhaps not surprisingly – the impact gets only worse with increased number of attack points.

More importantly, we have shown that the more practical and modern continual learning approaches are also extremely vulnerable to poisoning attacks. Specifically, we utilized the backdoor poisoning attack strategy to adversely impact the performance of continual learning (CL) algorithms. We showed that an attacker can take advantage of a CL algorithm’s very ability to continuously learn new information over time, and use that ability against itself by forcing to retain even the smallest amount of *misinformation* in the form of adversarial backdoor pattern. We hence showed that, just like human brain can be forced to learn misinformation while learning sequentially, so can CL models when tasked to learn under continual learning settings. To make the attack more stealthier, our attack strategy uses a completely *imperceptible* backdoor pattern and provides it to only 1% of the training data of any of its chosen single task. Even with this small amount of

imperceptible misinformation, the attacker can easily induce false learning with pin-point targeted damage, and force the CL model to forget any task and misclassify the instances of any class as any other class of the attacker's choosing.

To defend such a serious and insidious imperceptible backdoor attack to CL models, a novel defensive framework is proposed in this work. We term our defensive framework as adversary aware continual learning (AACL). The framework similar to the imperceptible backdoor attacks, uses a small amount of defensive (decoy) samples to force the continual learning (CL) models to learn a pattern, more specifically, a fixed defensive pattern, which is i) entirely different than the attacker's unknown imperceptible pattern and, ii) stronger (perceptible) than the attacker's pattern. While the attacker's goal is to associate its imperceptible pattern with its chosen false target label, the defender's goal is to remove that association using an overpowering defensive pattern. After training, with an image including both patterns presented to the model in a given test sample, the model pays more attention to the core features of the original image along with the fixed defense pattern to make its decision, thereby ignoring the attacker's pattern and thus correctly classifying the test sample.

We demonstrated the success of our proposed defensive framework using various continual learning algorithms and datasets. Moreover, we showed that our proposed framework can defend the continual learning algorithms regardless of which task or what time step the attack happens. We believe that this is our first critical step to ensure robustness in practical continual learning algorithm, which is of paramount importance to achieve the goal of artificial general intelligence (AGI).

To summarize, the main contributions of this dissertation are as follows

- We proposed a poisoning attack strategy to adversely impact the performance of covariate shift adaptation approaches.
- We utilized the backdoor poisoning attack strategy to implant small amount of misinformation in the form imperceptible backdoor pattern in the continual learning mod-

els. The misinformation is used to cause the targeted misclassification at test time.

- We proposed a novel defensive framework employing a defensive (decoy) samples to mitigate the impact of the imperceptible backdoor attacks in the CL model. We called our framework as Adversary Aware Continual Learning (AACL).

7.1 Summary of Future Work

We envision our future work to cover the following aspects

1. We will propose an optimization strategy to find a defensive pattern that is optimal for the specific model under attack and optimally defend it against any adversarial attacks.
2. We will explore different strategies to equally improve the robust (defense) performance on all the tasks. Note that ideally we want to achieve a robust (defense) performance to become exactly equal to the clean performance on a given task.
3. Other forms of adversarial attacks needs to be evaluated against CL approaches for instance clean label imperceptible backdoor attacks need to be designed to obtain an even stealthier attack setting. Generalized defensive mechanism also needs to be proposed that can defend not only imperceptible backdoor attacks proposed in this work but also any type of threat to the CL models.

References

- [1] M. De Lange *et al.*, “Continual learning: A comparative study on how to defy forgetting in classification tasks,” *arXiv preprint arXiv:1909.08383*, 2019.
- [2] M. McCloskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” in *Psychology of learning and motivation*, vol. 24, Elsevier, 1989, pp. 109–165.
- [3] S. Grossberg, “Nonlinear neural networks: Principles, mechanisms, and architectures,” *Neural networks*, vol. 1, no. 1, pp. 17–61, 1988.
- [4] R. M. French, “Catastrophic forgetting in connectionist networks,” *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.
- [5] R. Ratcliff, “Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions,” *Psychological review*, vol. 97, no. 2, p. 285, 1990.
- [6] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, “Dark experience for general continual learning: A strong, simple baseline,” *ArXiv*, vol. abs/2004.07211, 2020.
- [7] R. Aljundi, “Continual learning in neural networks,” *arXiv preprint arXiv:1910.02718*, 2019.
- [8] H. Shin, J. K. Lee, J. Kim, and J. Kim, “Continual learning with deep generative replay,” in *Advances in Neural Information Processing Systems*, 2017, pp. 2990–2999.
- [9] X. Liu *et al.*, “Generative feature replay for class-incremental learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 226–227.
- [10] G. Shen, S. Zhang, X. Chen, and Z.-H. Deng, “Generative feature replay with orthogonal weight modification for continual learning,” *arXiv preprint arXiv:2005.03490*, 2020.
- [11] G. M. van de Ven and A. S. Tolias, “Three scenarios for continual learning,” *arXiv preprint arXiv:1904.07734*, 2019.
- [12] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa, “Visual domain adaptation: A survey of recent advances,” *IEEE signal processing magazine*, vol. 32, no. 3, pp. 53–69, 2015.

- [13] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.
- [14] A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal,” *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.
- [15] D. L. Silver and R. E. Mercer, “The task rehearsal method of life-long learning: Overcoming impoverished data,” in *Conference of the Canadian Society for Computational Studies of Intelligence*, Springer, 2002, pp. 90–101.
- [16] J. Rueckl, “Jumpnet: A multiple-memory connectionist architecture,” in *Proceedings of the 15 th Annual Conference of the Cognitive Science Society*, vol. 24, 1993, pp. 866–871.
- [17] B. Ans and S. Rousset, “Avoiding catastrophic forgetting by coupling two reverberating neural networks,” *Comptes Rendus de l’Académie des Sciences-Series III-Sciences de la Vie*, vol. 320, no. 12, pp. 989–997, 1997.
- [18] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, “Learn++: An incremental learning algorithm for supervised neural networks,” *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.
- [19] R. M. French, “Semi-distributed representations and catastrophic forgetting in connectionist networks,” *Connection Science*, vol. 4, no. 3-4, pp. 365–377, 1992.
- [20] R. M. Nosofsky, J. K. Kruschke, and S. C. McKinley, “Combining exemplar-based category representations and connectionist learning rules,” *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 18, no. 2, p. 211, 1992.
- [21] D. E. Rumelhart, “Reducing interference in distributed memories through episodic gating,” *From learning theory to connectionist theory*, vol. 1, p. 227, 1992.
- [22] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, “An empirical investigation of catastrophic forgetting in gradient-based neural networks,” *arXiv preprint arXiv:1312.6211*, 2013.
- [23] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Max-out networks,” in *International conference on machine learning*, PMLR, 2013, pp. 1319–1327.
- [24] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber, “Compete to compute,” *Advances in neural information processing systems*, vol. 26, 2013.

- [25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [26] A. A. Rusu *et al.*, “Progressive neural networks,” *arXiv preprint arXiv:1606.04671*, 2016.
- [27] T. Lesort, “Continual learning: Tackling catastrophic forgetting in deep neural networks with replay processes,” *arXiv preprint arXiv:2007.00487*, 2020.
- [28] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, “Lifelong learning with dynamically expandable networks,” *arXiv preprint arXiv:1708.01547*, 2017.
- [29] Z. Wang *et al.*, “Learning to prompt for continual learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 139–149.
- [30] A. Mallya, D. Davis, and S. Lazebnik, “Piggyback: Adapting a single network to multiple tasks by learning to mask weights,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 67–82.
- [31] A. Mallya and S. Lazebnik, “Packnet: Adding multiple tasks to a single network by iterative pruning,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7765–7773.
- [32] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, “Overcoming catastrophic forgetting with hard attention to the task,” in *International Conference on Machine Learning*, PMLR, 2018, pp. 4548–4557.
- [33] C. Fernando *et al.*, “Pathnet: Evolution channels gradient descent in super neural networks,” *arXiv preprint arXiv:1701.08734*, 2017.
- [34] Z. Li and D. Hoiem, “Learning without forgetting,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.
- [35] G. Hinton, O. Vinyals, J. Dean, *et al.*, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.
- [36] J. Kirkpatrick *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.
- [37] J. Schwarz *et al.*, “Progress & compress: A scalable framework for continual learning,” *arXiv preprint arXiv:1805.06370*, 2018.

- [38] F. Zenke, B. Poole, and S. Ganguli, “Continual learning through synaptic intelligence,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 3987–3995.
- [39] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, “Memory aware synapses: Learning what (not) to forget,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 139–154.
- [40] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, “Overcoming catastrophic forgetting by incremental moment matching,” *Advances in neural information processing systems*, vol. 30, 2017.
- [41] X. He and H. Jaeger, “Overcoming catastrophic interference using conceptor-aided backpropagation,” in *International Conference on Learning Representations*, 2018.
- [42] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “Icarl: Incremental classifier and representation learning,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.
- [43] A. Chaudhry *et al.*, “Continual learning with tiny episodic memories,” 2019.
- [44] C. Atkinson, B. McCane, L. Szymanski, and A. Robins, “Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks,” *arXiv preprint arXiv:1802.03875*, 2018.
- [45] D. Lopez-Paz and M. Ranzato, “Gradient episodic memory for continual learning,” *Advances in neural information processing systems*, vol. 30, 2017.
- [46] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, “Efficient lifelong learning with a-gem,” *arXiv preprint arXiv:1812.00420*, 2018.
- [47] J. Rajasegaran, M. Hayat, S. H. Khan, F. S. Khan, and L. Shao, “Random path selection for continual learning,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [48] J. Rajasegaran, S. Khan, M. Hayat, F. S. Khan, and M. Shah, “Itaml: An incremental task-agnostic meta-learning approach,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 588–13 597.
- [49] M. De Lange *et al.*, “A continual learning survey: Defying forgetting in classification tasks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3366–3385, 2021.
- [50] I. Goodfellow *et al.*, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

- [51] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [52] G. M. van de Ven and A. S. Tolias, “Generative replay with feedback connections as a general strategy for continual learning,” *arXiv preprint arXiv:1809.10635*, 2018.
- [53] F. Lavda, J. Ramapuram, M. Gregorova, and A. Kalousis, “Continual classification learning using generative models,” *arXiv preprint arXiv:1810.10612*, 2018.
- [54] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, “Adversarial machine learning,” in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, ACM, 2011, pp. 43–58.
- [55] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” *arXiv preprint arXiv:1206.6389*, 2012.
- [56] B. Biggio *et al.*, “Evasion attacks against machine learning at test time,” in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2013, pp. 387–402.
- [57] R. S. S. Kumar *et al.*, “Adversarial machine learning-industry perspectives,” in *2020 IEEE security and privacy workshops (SPW)*, IEEE, 2020, pp. 69–75.
- [58] B. Biggio, I. Pillai, S. Rota Bulò, D. Ariu, M. Pelillo, and F. Roli, “Is data clustering in adversarial settings secure?” In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, 2013, pp. 87–98.
- [59] B. I. Rubinstein *et al.*, “Antidote: Understanding and defending against poisoning of anomaly detectors,” in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 2009, pp. 1–14.
- [60] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, “Is feature selection secure against training data poisoning?” In *International Conference on Machine Learning*, 2015, pp. 1689–1698.
- [61] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [62] T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” *arXiv preprint arXiv:1708.06733*, 2017.
- [63] A. Shafahi *et al.*, “Poison frogs! targeted clean-label poisoning attacks on neural networks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

- [64] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [65] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 2938–2948.
- [66] B. Tran, J. Li, and A. Madry, “Spectral signatures in backdoor attacks,” *Advances in neural information processing systems*, vol. 31, 2018.
- [67] B. Chen *et al.*, “Detecting backdoor attacks on deep neural networks by activation clustering,” in *SafeAI@ AAAI*, 2019.
- [68] A. Chan and Y.-S. Ong, “Poison as a cure: Detecting & neutralizing variable-sized backdoor attacks in deep neural networks,” *arXiv preprint arXiv:1911.08040*, 2019.
- [69] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, “Strip: A defence against trojan attacks on deep neural networks,” in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 113–125.
- [70] S. Udeshi, S. Peng, G. Woo, L. Loh, L. Rawshan, and S. Chattopadhyay, “Model agnostic defence against backdoor attacks in machine learning,” *IEEE Transactions on Reliability*, 2022.
- [71] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, and S. Xia, “Rethinking the trigger of backdoor attack,” *arXiv preprint arXiv:2004.04692*, 2020.
- [72] B. G. Doan, E. Abbasnejad, and D. C. Ranasinghe, “Februus: Input purification defense against trojan attacks on deep neural network systems,” in *Annual Computer Security Applications Conference*, 2020, pp. 897–912.
- [73] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, “Grad-cam: Why did you say that?” *arXiv preprint arXiv:1611.07450*, 2016.
- [74] K. Liu, B. Dolan-Gavitt, and S. Garg, “Fine-pruning: Defending against backdoor- ing attacks on deep neural networks,” in *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer, 2018, pp. 273–294.
- [75] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, “Abs: Scanning neural networks for back-doors by artificial brain stimulation,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1265–1282.

- [76] E. Sarkar, Y. Alkindi, and M. Maniatakos, “Backdoor suppression in neural networks using input fuzzing and majority voting,” *IEEE Design & Test*, vol. 37, no. 2, pp. 103–110, 2020.
- [77] B. Wang *et al.*, “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 707–723.
- [78] C. Szegedy *et al.*, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.
- [79] M. Wang and W. Deng, “Deep visual domain adaptation: A survey,” *Neurocomputing*, vol. 312, pp. 135–153, 2018.
- [80] M. Umer, C. Frederickson, and R. Polikar, “Vulnerability of covariate shift adaptation against malicious poisoning attacks,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.
- [81] M. Umer, C. Frederickson, and R. Polikar, “Adversarial poisoning of importance weighting in domain adaptation,” in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2018, pp. 381–388.
- [82] K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf, and A. J. Smola, “Integrating structured biological data by kernel maximum mean discrepancy,” *Bioinformatics*, vol. 22, no. 14, e49–e57, 2006.
- [83] H. Hachiya, T. Akiyama, M. Sugiyama, and J. Peters, “Adaptive importance sampling with automatic model selection in value function approximation,” in *AAAI*, 2008, pp. 1351–1356.
- [84] S. Bickel and T. Scheffer, “Dirichlet-enhanced spam filtering based on biased samples,” in *Advances in neural information processing systems*, 2007, pp. 161–168.
- [85] M. Sugiyama, M. Krauledat, and K.-R. Mäñller, “Covariate shift adaptation by importance weighted cross validation,” *Journal of Machine Learning Research*, vol. 8, no. May, pp. 985–1005, 2007.
- [86] J. Heckman *et al.*, “Sample selection bias as a specification error,” *Applied Econometrics*, vol. 31, no. 3, pp. 129–137, 2013.
- [87] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. The MIT Press, 2009.

- [88] H. Shimodaira, “Improving predictive inference under covariate shift by weighting the log-likelihood function,” *Journal of statistical planning and inference*, vol. 90, no. 2, pp. 227–244, 2000.
- [89] J. Huang, A. Gretton, K. M. Borgwardt, B. Schölkopf, and A. J. Smola, “Correcting sample selection bias by unlabeled data,” in *Advances in neural information processing systems*, 2006, pp. 601–608.
- [90] S. Bickel, M. Brückner, and T. Scheffer, “Discriminative learning under covariate shift,” *Journal of Machine Learning Research*, vol. 10, no. Sep, pp. 2137–2155, 2009.
- [91] M. Sugiyama, S. Nakajima, H. Kashima, P. V. Buenau, and M. Kawanabe, “Direct importance estimation with model selection and its application to covariate shift adaptation,” in *Advances in neural information processing systems*, 2008, pp. 1433–1440.
- [92] T. Kanamori, S. Hido, and M. Sugiyama, “A least-squares approach to direct importance estimation,” *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1391–1445, 2009.
- [93] H. Hachiya, M. Sugiyama, and N. Ueda, “Importance-weighted least-squares probabilistic classifier for covariate shift adaptation with application to human activity recognition,” *Neurocomputing*, vol. 80, pp. 93–101, 2012.
- [94] M. Umer, R. Polikar, and C. Frederickson, “Level iw: Learning extreme verification latency with importance weighting,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 1740–1747.
- [95] M. Umer and R. Polikar, “Adversarial targeted forgetting in regularization and generative based continual learning models,” in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.
- [96] M. Umer, G. Dawson, and R. Polikar, “Targeted forgetting and false memory formation in continual learners through adversarial backdoor attacks,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.
- [97] R. M. Nichols and E. F. Loftus, “Who is susceptible in three false memory tasks?” *Memory*, vol. 27, no. 7, pp. 962–984, 2019, PMID: 31046606. DOI: 10.1080/09658211.2019.1611862. eprint: <https://doi.org/10.1080/09658211.2019.1611862>. [Online]. Available: <https://doi.org/10.1080/09658211.2019.1611862>.
- [98] S. J. Frenda, R. M. Nichols, and E. F. Loftus, “Current issues and advances in misinformation research,” *Current Directions in Psychological Science*, vol. 20, no. 1, pp. 20–23, 2011.

- [99] S. Ramirez *et al.*, “Creating a false memory in the hippocampus,” *Science*, vol. 341, no. 6144, pp. 387–391, 2013.
- [100] K. Deisseroth, “Optogenetics,” *Nature methods*, vol. 8, no. 1, pp. 26–29, 2011.
- [101] T. Lesort, H. Caselles-Dupré, M. Garcia-Ortiz, A. Stoian, and D. Filliat, “Generative models from the perspective of continual learning,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.
- [102] G. M. Van de Ven, H. T. Siegelmann, and A. S. Tolias, “Brain-inspired replay for continual learning with artificial neural networks,” *Nature communications*, vol. 11, no. 1, p. 4069, 2020.
- [103] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, “Recent advances in adversarial training for adversarial robustness,”
- [104] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against deep learning systems using adversarial examples,” *arXiv preprint arXiv:1602.02697*, vol. 1, no. 2, p. 3, 2016.
- [105] N. Papernot, P. McDaniel, and I. Goodfellow, “Transferability in machine learning: From phenomena to black-box attacks using adversarial samples,” *arXiv preprint arXiv:1605.07277*, 2016.
- [106] J. Zhang and C. Li, “Adversarial examples: Opportunities and challenges,” *IEEE transactions on neural networks and learning systems*, vol. 31, no. 7, pp. 2578–2593, 2019.
- [107] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song, “Natural adversarial examples,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 262–15 271.
- [108] Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels,” *Advances in neural information processing systems*, vol. 31, 2018.
- [109] M. Martinez and R. Stiefelhagen, “Taming the cross entropy loss,” in *German Conference on Pattern Recognition*, Springer, 2018, pp. 628–637.
- [110] H. Song, M. Kim, D. Park, Y. Shin, and J.-G. Lee, “Learning from noisy labels with deep neural networks: A survey,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
- [111] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, “Learning with noisy labels,” *Advances in neural information processing systems*, vol. 26, 2013.

- [112] G. Dawson and R. Polikar, “Rethinking noisy label models: Labeler-dependent noise with adversarial awareness,” *arXiv preprint arXiv:2105.14083*, 2021.
- [113] D. Zhang, T. Zhang, Y. Lu, Z. Zhu, and B. Dong, “You only propagate once: Accelerating adversarial training via maximal principle,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [114] A. Shafahi *et al.*, “Adversarial training for free!” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [115] E. Wong, L. Rice, and J. Z. Kolter, “Fast is better than free: Revisiting adversarial training,” *arXiv preprint arXiv:2001.03994*, 2020.
- [116] V. M. Souza, D. F. Silva, J. Gama, and G. E. Batista, “Data stream classification guided by clustering on nonstationary environments and extreme verification latency,” in *Proceedings of the 2015 SIAM international conference on data mining*, SIAM, 2015, pp. 873–881.
- [117] M. Umer, C. Frederickson, and R. Polikar, “Learning under extreme verification latency quickly: Fast compose,” in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2016, pp. 1–8.
- [118] J. Hoffman, T. Darrell, and K. Saenko, “Continuous manifold based adaptation for evolving visual domains,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 867–874.
- [119] A. Raghunathan, S. M. Xie, F. Yang, J. C. Duchi, and P. Liang, “Adversarial training can hurt generalization,” *arXiv preprint arXiv:1906.06032*, 2019.
- [120] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, “Robustness may be at odds with accuracy,” in *International Conference on Learning Representations*, 2019.
- [121] H. Wang, T. Chen, S. Gui, T. Hu, J. Liu, and Z. Wang, “Once-for-all adversarial training: In-situ tradeoff between robustness and accuracy for free,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 7449–7461, 2020.

Appendix A

Derivations of Poisoning Importance Estimation

Derivation of Poisoning Logistic Regression Classifier Based Importance Estimation. The Logistic Regression classifier expresses the conditional probability $p(\eta|\mathbf{x})$ by employing a parametric model of the following form

$$p(\eta|\mathbf{x}) = \frac{1}{1 + e^{-(\boldsymbol{\theta}^T \mathbf{x} + b)}} = h_{\boldsymbol{\theta}, b}(\mathbf{x}) \quad (41)$$

where $\boldsymbol{\theta}, b$ are the parameters to be learned by minimizing the negative log-likelihood $\mathcal{L}(\boldsymbol{\theta}, b)$ given as

$$\mathcal{L}(\boldsymbol{\theta}, b) = - \sum_{i=1}^{n_{tr}} (\eta_i^{tr} \log(h_{\boldsymbol{\theta}, b}(\mathbf{x}_i^{tr})) + (1 - \eta_i^{tr}) \log(1 - h_{\boldsymbol{\theta}, b}(\mathbf{x}_i^{tr}))) \quad (42)$$

where η_i^{tr} is the corresponding label for an i^{th} training instance \mathbf{x}_i^{tr} . The attacker's goal is to maximize the loss function with respect to the attack samples under the constraint that the original parameters of the model are still obtained by minimizing the actual loss function. The attacker's goal then can be characterized in terms of the objective function \mathcal{W} as shown below

$$\begin{aligned} \max_{\mathbf{x}_c} \mathcal{W} &= \mathcal{L}(\boldsymbol{\theta}, b) \\ \text{subject to } \boldsymbol{\theta}, b &= \arg \min_{\boldsymbol{\theta}, b} \mathcal{L}(\boldsymbol{\theta}, b) \end{aligned} \quad (43)$$

Therefore, the attacker wants to calculate $\frac{\partial \mathcal{W}}{\partial \mathbf{x}_c}$ in order to achieve its objective, which we derive as follows

$$\begin{aligned}
\frac{\partial \mathcal{W}}{\partial \mathbf{x}_c} &= \frac{\partial}{\partial \mathbf{x}_c} \left[-\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (\eta_i^{tr} \log(h_{\theta,b}(\mathbf{x}_i^{tr})) + (1 - \eta_i^{tr}) \log(1 - h_{\theta,b}(\mathbf{x}_i^{tr}))) \right] \\
&= -\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \left[\left(\frac{\eta_i^{tr}}{h_{\theta,b}(\mathbf{x}_i^{tr})} - \frac{1 - \eta_i^{tr}}{1 - h_{\theta,b}(\mathbf{x}_i^{tr})} \right) (h_{\theta,b}(\mathbf{x}_i^{tr}) (1 - h_{\theta,b}(\mathbf{x}_i^{tr})) \frac{\partial}{\partial \mathbf{x}_c} ((\boldsymbol{\theta}^T \mathbf{x}_i + b))) \right] \\
&= -\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} [(\eta_i^{tr} - h_{\theta,b}(\mathbf{x}_i^{tr})) (\boldsymbol{\theta}^T \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_c} + \mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c})] \\
&= \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (h_{\theta,b}(\mathbf{x}_i) - \eta_i) (\mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c})
\end{aligned} \tag{44}$$

The formula shown above is not simple to compute because we do not know $\frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c}$ and $\frac{\partial b}{\partial \mathbf{x}_c}$. In order to find these, we use the Karush-Kuhn-Tucker (KKT) condition of the inner optimization problem. The KKT stability condition of logistic regression (LR) classifier states that at optimal \mathbf{x}_c , the objective function $\mathcal{L}(\boldsymbol{\theta}, b)$ is stable, i.e. the gradient of $\mathcal{L}(\boldsymbol{\theta}, b)$ with respect to $\boldsymbol{\theta}$ and b is zero. Now following the approach described by Biggio et al. in [55], an assumption is made here that states that *the KKT condition under perturbation of \mathbf{x}_c remains satisfied*. Since LR classifier tries to optimize $\mathcal{L}(\boldsymbol{\theta}, b)$ for any input value of \mathbf{x}_c , it is sensible to assume that $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \mathcal{L}}{\partial b}$ remain equal to zero after a small change in the value of \mathbf{x}_c . Having made that assumption, the followings are true

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right)^T &= \mathbf{0} \\
\frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial b} \right) &= 0
\end{aligned} \tag{45}$$

Since we know that

$$\begin{aligned}
\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right)^T &= \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (h_{\theta,b}(\mathbf{x}_i) - \eta_i) \mathbf{x}_i \\
\frac{\partial \mathcal{L}}{\partial b} &= \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (h_{\theta,b}(\mathbf{x}_i) - \eta_i)
\end{aligned} \tag{46}$$

which implies that

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right)^T &= \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \left[\mathbf{x}_i \frac{\partial}{\partial \mathbf{x}_c} (h_{\boldsymbol{\theta},b}(\mathbf{x}_i) - \eta_i) + (h_{\boldsymbol{\theta},b}(\mathbf{x}_i) - \eta_i) \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_c} \right] = \\ &= \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \left[\mathbf{x}_i \frac{\partial}{\partial \mathbf{x}_c} (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) + (h_{\boldsymbol{\theta},b}(\mathbf{x}_i) - \eta_i) \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_c} \right] \end{aligned} \quad (47)$$

Now, given that the attack point \mathbf{x}_c is added into the training data, we have:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right)^T &= \frac{1}{n} \left[\sum_{i=1, i \neq \mathbf{x}_c}^n \mathbf{x}_i (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) \frac{\partial}{\partial \mathbf{x}_c} (\boldsymbol{\theta}^T \mathbf{x}_i + b) \right. \\ &\quad \left. + \mathbf{x}_c (h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))) \frac{\partial}{\partial \mathbf{x}_c} (\boldsymbol{\theta}^T \mathbf{x}_c + b) \right] \\ &\quad + \frac{1}{n} \left[\sum_{i=1, i \neq \mathbf{x}_c}^n (h_{\boldsymbol{\theta},b}(\mathbf{x}_i) - \eta_i) \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_c} + (h_{\boldsymbol{\theta},b}(\mathbf{x}_c) - \eta_c) \frac{\partial \mathbf{x}_c}{\partial \mathbf{x}_c} \right] \\ &= \frac{1}{n} \left[\sum_{i=1, i \neq \mathbf{x}_c}^n \mathbf{x}_i (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) (\boldsymbol{\theta}^T \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_c} + \mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c}) \right. \\ &\quad \left. + \mathbf{x}_c (h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))) (\boldsymbol{\theta}^T \frac{\partial \mathbf{x}_c}{\partial \mathbf{x}_c} + \mathbf{x}_c^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c}) \right] + \frac{1}{n} [(h_{\boldsymbol{\theta},b}(\mathbf{x}_c) - \eta_c) \mathbf{I}] \quad (48) \\ &= \frac{1}{n} \left[\sum_{i=1, i \neq \mathbf{x}_c}^n \mathbf{x}_i (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) (\mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c}) \right. \\ &\quad \left. + \mathbf{x}_c (h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))) (\boldsymbol{\theta}^T + \mathbf{x}_c^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{\partial b}{\partial \mathbf{x}_c}) \right] + \frac{1}{n} \left[\sum_{i=1}^n ((h_{\boldsymbol{\theta},b}(\mathbf{x}_c) - \eta_c) \mathbf{I}) \right] \\ &= \frac{1}{n} \sum_{i=1}^n (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) \mathbf{x}_i \mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{1}{n} \sum_{i=1}^n (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) \mathbf{x}_i \frac{\partial b}{\partial \mathbf{x}_c} \\ &\quad + \frac{1}{n} (h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))) (\mathbf{x}_c \boldsymbol{\theta}^T) + \frac{1}{n} \left[\sum_{i=1}^n ((h_{\boldsymbol{\theta},b}(\mathbf{x}_c) - \eta_c) \mathbf{I}) \right] \end{aligned}$$

where $n = n_{tr} + 1$: total number of training instances with one attack point \mathbf{x}_c added into it, and \mathbf{I} represents an identity matrix. Following the same procedure for solving $\frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial b} \right)$, we get the following

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \mathcal{L}}{\partial b} \right) &= \frac{1}{n} \sum_{i=1}^n (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) \mathbf{x}_i^T \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{x}_c} + \frac{1}{n} \sum_{i=1}^n (h_{\boldsymbol{\theta},b}(\mathbf{x}_i)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_i))) \frac{\partial b}{\partial \mathbf{x}_c} \\ &\quad + \frac{1}{n} (h_{\boldsymbol{\theta},b}(\mathbf{x}_c)) (1 - (h_{\boldsymbol{\theta},b}(\mathbf{x}_c))) (\boldsymbol{\theta}^T) \end{aligned} \quad (49)$$

If we set the derivatives obtained in Equation 48 and Equation 49 to zero and rearrange the resulting equations in matrix form, we get the following linear system

$$\begin{bmatrix} \Sigma & \mu \\ \mu^T & \beta \end{bmatrix} \begin{bmatrix} \frac{\partial \theta}{\partial \mathbf{x}_c} \\ \frac{\partial b^T}{\partial \mathbf{x}_c} \end{bmatrix} = -\frac{1}{n} \begin{bmatrix} M \\ r \end{bmatrix} \quad (50)$$

where

$$\Sigma = \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i) \mathbf{x}_i \mathbf{x}_i^T$$

$$\mu = \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i) \mathbf{x}_i$$

$$\beta = \frac{1}{n} K(\mathbf{x}_c)$$

$$M = (h_{\theta,b}(\mathbf{x}_c) - \eta_c) \mathbf{I} + (h_{\theta,b}(\mathbf{x}_c))(1 - h_{\theta,b}(\mathbf{x}_c)) \mathbf{x}_c \boldsymbol{\theta}^T$$

$$r = (h_{\theta,b}(\mathbf{x}_c))(1 - h_{\theta,b}(\mathbf{x}_c)) \boldsymbol{\theta}^T$$

and $K(\mathbf{x}) = (h_{\theta,b}(\mathbf{x}))(1 - h_{\theta,b}(\mathbf{x}))$. The derivatives $\frac{\partial \theta}{\partial \mathbf{x}_c}$ and $\frac{\partial b}{\partial \mathbf{x}_c}$ can be finally obtained by solving the linear system given by Equation 50.

Derivation for Poisoning Unconstrained Least Squares Importance Estimation. Recall from chapter 3 that unconstrained least squares importance estimation models the importance ratio $w(\mathbf{x})$ through a linear-in-parameter model as

$$\hat{w}(\mathbf{x}) = \sum_{l=1}^t \alpha_l \phi_l(\mathbf{x}) \quad (51)$$

where t is the number of parameters, $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_t)$ are the parameters to be learned from the data samples, and $\{\phi_l(\mathbf{x})\}_{l=1}^t$ are the basis functions such that $\phi_l(\mathbf{x}) > 0 \forall \mathbf{x}$. The parameters α are determined by minimizing the following objective function

$$\hat{J}(\alpha) = \frac{1}{2} \alpha^T \hat{\mathbf{H}} \alpha - \hat{\mathbf{h}}^T \alpha + \frac{\lambda}{2} \alpha^T \alpha \quad (52)$$

where, $\hat{\mathbf{H}}$ is a $t \times t$ matrix whose elements are $\hat{H}_{l,l'} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \phi_l(\mathbf{x}_i^{tr}) \phi_{l'}(\mathbf{x}_i^{tr})$, and $\hat{\mathbf{h}}$ is a t -dimensional vector with elements $\hat{h}_l = \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \phi_l(\mathbf{x}_j^{te})$. Note that a quadratic penalty term $\frac{\lambda}{2} \boldsymbol{\alpha}^T \boldsymbol{\alpha}$ is added to the objective function for regularization purposes. The goal of an attacker is then to maximize the objective function defined in Equation 52 with respect to the attack point under the constraint that the parameters of the linear model are still obtained by minimizing the same objective function. Mathematically, the goal of an attacker can be represented as a bi-level optimization problem shown below:

$$\begin{aligned} \mathcal{W} &= \max_{\mathbf{x}_c} \hat{J}(\boldsymbol{\alpha}) \\ \text{subject to } \boldsymbol{\alpha} &= \arg \min_{\boldsymbol{\alpha}} \hat{J}(\boldsymbol{\alpha}) \end{aligned} \quad (53)$$

The partial derivative of \mathcal{W} with respect to the attack points \mathbf{x}_c using multivariate chain rule is as follows

$$\begin{aligned} \frac{\partial \mathcal{W}}{\partial \mathbf{x}_c} &= (\hat{\mathbf{H}} \boldsymbol{\alpha})^T \frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} - \hat{\mathbf{h}}^T \frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} + \lambda \boldsymbol{\alpha}^T \frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} + \\ &\quad \frac{1}{2} \boldsymbol{\alpha}^T \left(\frac{\partial \hat{\mathbf{H}}}{\partial \mathbf{x}_c} \right) \boldsymbol{\alpha}. \end{aligned} \quad (54)$$

$\partial \hat{\mathbf{H}} / \partial \mathbf{x}_c$ and $\partial \boldsymbol{\alpha} / \partial \mathbf{x}_c$ in Equation 54 need to be known in order to compute $\frac{\partial \mathcal{W}}{\partial \mathbf{x}_c}$. To compute these, we use KKT stability conditions of the inner optimization problem, which states that

$$\frac{\partial}{\partial \mathbf{x}_c} \left(\frac{\partial \hat{J}(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} \right) = \mathbf{0} \quad (55)$$

Starting with Equation 19, and computing the derivative in Equation 55, we arrive at

$$\frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{x}_c} = -[\hat{\mathbf{H}} + \lambda \mathbf{I}]^{-1} \frac{\partial \hat{\mathbf{H}}}{\partial \mathbf{x}_c} \boldsymbol{\alpha} \quad (56)$$

Equation 56 also requires computing $\partial \hat{\mathbf{H}} / \partial \mathbf{x}_c$. As mentioned previously that $\hat{\mathbf{H}}$ is $t \times t$ matrix with $\hat{H}_{l,l'} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \phi_l(\mathbf{x}_i^{tr}) \phi_{l'}(\mathbf{x}_i^{tr})$. We pick the Gaussian kernel as our basis func-

tions ϕ , leading us to the following definition for \hat{H}

$$\hat{H}_{l,l'} = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \exp\left(-\frac{\|\mathbf{x}_i^{tr} - \mathbf{c}_l\|^2 + \|\mathbf{x}_i^{tr} - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) \quad (57)$$

for $l, l' = 1, \dots, t$

where, σ denotes the bandwidth of the Gaussian kernel, $\mathbf{c}_l, \mathbf{c}_{l'}$ denotes the t kernel centers picked randomly from test data instances, and \mathbf{x}_i^{tr} represents the n_{tr} total training instances. Now, let's assume an attacker inserts a single attack point \mathbf{x}_c into the training data which will cause the matrix \hat{H} to be redefined as follows

$$\hat{H}_{l,l'} = \frac{1}{n_{tr} + 1} \left[\sum_{i=1, i \neq x_c}^{n_{tr}} \exp\left(-\frac{\|\mathbf{x}_i^{tr} - \mathbf{c}_l\|^2 + \|\mathbf{x}_i^{tr} - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) + \exp\left(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}\right) \right] \quad (58)$$

for $l, l' = 1, \dots, t$

Equation 58 defines a matrix \hat{H} when a single attack point \mathbf{x}_c is added to the training data. Now the derivative of $\hat{H}_{l,l'}$ w.r.t the attack point \mathbf{x}_c will cause the first summation term in the above expression to go to zero as it does not involve \mathbf{x}_c and we will have the

following expression

$$\begin{aligned}
\frac{\partial \hat{H}_{l,l'}}{\partial \mathbf{x}_c} &= \frac{1}{n_{tr} + 1} \frac{\partial}{\partial \mathbf{x}_c} [\exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2})] \\
&= \frac{-1}{2\sigma^2(n_{tr} + 1)} \exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}) \\
&\quad \frac{\partial}{\partial \mathbf{x}_c} [(\mathbf{x}_c - \mathbf{c}_l)^T (\mathbf{x}_c - \mathbf{c}_l) + (\mathbf{x}_c - \mathbf{c}_{l'})^T (\mathbf{x}_c - \mathbf{c}_{l'})] \\
&= \frac{-1}{2\sigma^2(n_{tr} + 1)} \exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}) \\
&\quad \frac{\partial}{\partial \mathbf{x}_c} [\mathbf{x}_c^T \mathbf{x}_c - \mathbf{x}_c^T \mathbf{c}_l - \mathbf{c}_l^T \mathbf{x}_c + \mathbf{c}_l^T \mathbf{c}_l + \mathbf{x}_c^T \mathbf{x}_c - \mathbf{x}_c^T \mathbf{c}_{l'} - \mathbf{c}_{l'}^T \mathbf{x}_c + \mathbf{c}_{l'}^T \mathbf{c}_{l'}] \quad (59) \\
&= \frac{-1}{2\sigma^2(n_{tr} + 1)} \exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}) \\
&\quad \frac{\partial}{\partial \mathbf{x}_c} [2\mathbf{x}_c^T \mathbf{x}_c - 2\mathbf{x}_c^T \mathbf{c}_l + \mathbf{c}_l^T \mathbf{c}_l - 2\mathbf{x}_c^T \mathbf{c}_{l'} + \mathbf{c}_{l'}^T \mathbf{c}_{l'}] \\
&= \frac{-1}{2\sigma^2(n_{tr} + 1)} \exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}) [4\mathbf{x}_c - 2\mathbf{c}_l - 2\mathbf{c}_{l'}] \\
&= \frac{-1}{\sigma^2(n_{tr} + 1)} \exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}) [2\mathbf{x}_c - \mathbf{c}_l - \mathbf{c}_{l'}]
\end{aligned}$$

So, finally we have the following

$$\frac{\partial \hat{H}_{l,l'}}{\partial \mathbf{x}_c} = \frac{-1}{\sigma^2(n_{tr} + 1)} \exp(-\frac{\|\mathbf{x}_c - \mathbf{c}_l\|^2 + \|\mathbf{x}_c - \mathbf{c}_{l'}\|^2}{2\sigma^2}) [2\mathbf{x}_c - \mathbf{c}_l - \mathbf{c}_{l'}] \quad (60)$$

where $l, l' = 1, 2, \dots, t$. If we put the above expression into Equation 56, we can finally find the required Equation 54 to compute $\frac{\partial \mathcal{W}}{\partial \mathbf{x}_c}$ in order to find the optimal attack point \mathbf{x}_c .

Appendix B

Attacking Regularization Based Continual Learning (CL) Approaches

Regularization based CL approaches are useful as they neither store data from the previous tasks nor add more layers or nodes to the network with each new incoming task, and thus avoid data storage and architectural complexity issues. However, with a fixed capacity single network and with no access to previous data at all – not even in the pseudo form – these approaches do struggle for challenging datasets under continual learning settings. More specifically, regularization-based approaches only perform well for continual variant of MNIST dataset and fail for more complex and realistic datasets such as continual variant of CIFAR-10, and CIFAR-100.

We evaluate our proposed attack strategy to attack regularization based continual learning approaches and demonstrate that these approaches although not practical, however, for those cases where they show reasonable performance, their test time performance can be strongly impaired by an intelligent attacker.

To attack regularization based approaches using MNIST based continual dataset, we assume without any loss of generality that the target task of the attacker is Task 1, and the attacker’s desired false label is class 0. In this setting, we create a small number of malicious samples by inserting an imperceptible backdoor pattern into only 1% of the training data and assigning them a false label of 0. The falsely-labeled malicious samples (containing the imperceptible backdoor pattern) are then added to the clean training data, and the model is trained using regularization-based continual learning (CL) algorithms.

At inference time, we compute test accuracy for each task when there is “no attack” and when there is an “attack”. Note that we consider the two new attack scenarios; First scenario (worst-case) considers a setting where the attacker provides the attack or malicious backdoor samples in to the training data of every tasks except the attacker’s targeted first

task; Second scenario (realistic) considers a case where the attacker provides the malicious backdoor samples in to the training data of the single task more specifically in the last task in our case.

Recall that the attacker’s objective is to compromise Task 1 at test time; hence, in all tasks *except* the first task, test examples are presented to the model *without* the backdoor pattern, in order to monitor the model’s ability to retain performance on untargeted tasks. For the targeted task, 100% of the test examples from each class – except class 0 – are presented to the model with imperceptible backdoor tagged images. The attacker’s goal here is to have each of these tagged images to be misclassified as class 0. In other words, the attacker takes control of the classifier, by forcing it to classify any pattern it desires (by tagging it with an imperceptible backdoor image) to be misclassified as any class it wishes (class 0, in this example). The test accuracy for each task, as well as the average accuracy across all tasks, under no attack and with attack are presented in Table A1 for the first attack scenario and in Table A2 for the second attack scenario when attacking regularization-based CL approaches. All attack results are presented with mean and standard deviation computed over ten independent runs. Table A1 and Table A2 show that – in both attack scenarios– the malicious data with imperceptible backdoor pattern are sufficient to force all the regularization approaches to forget the first task, as the models’ 88-89% performance under no attack drops to about 10-14% when attacked for Task 1. Note that the drop in performance in the first attack scenario is more significant in the case of first attack setting, which is expected as the attack samples are provided in more tasks but most noteworthy the merely 1% of malicious samples provided in the only last task are more than enough to cause targeted forgetting on the attacker’s target task, i.e., Task 1. Also note that the performances on non-targeted tasks are not affected, as intended.

Table A1

Test Accuracy (in %) of Regularization-Based Continual Learning Approaches on MNIST with 1% Malicious Samples in Every Task except the First

Tasks	EWC		SI		Online EWC	
	Clean	Attack	Clean	Attack	Clean	Attack
Task 1	88.17	9.67 \pm 0.04	88.32	11.11 \pm 0.35	87.08	9.66 \pm 0.04
Task 2	96.37	96.27 \pm 0.18	95.13	95.28 \pm 0.72	95.08	96.08 \pm 0.32
Task 3	97.02	97.00 \pm 0.09	95.25	95.22 \pm 0.36	95.89	96.08 \pm 0.24
Task 4	96.74	96.79 \pm 0.07	94.31	94.03 \pm 0.41	95.75	95.29 \pm 0.25
Task 5	95.91	95.96 \pm 0.09	91.65	90.84 \pm 1.14	94.21	93.15 \pm 0.32

Table A2

Test Accuracy (in %) of Regularization-Based Continual Learning Approaches on MNIST with 1% Malicious Samples in only the Last Task

Tasks	EWC		SI		Online EWC	
	Clean	Attack	Clean	Attack	Clean	Attack
Task 1	88.17	10.01 \pm 0.20	88.32	14.11 \pm 0.53	87.08	11.25 \pm 0.27
Task 2	96.37	96.27 \pm 0.03	95.13	96.08 \pm 0.08	95.08	95.98 \pm 0.10
Task 3	97.02	97.00 \pm 0.03	95.25	95.22 \pm 0.06	95.89	96.08 \pm 0.05
Task 4	96.74	96.79 \pm 0.07	94.31	94.03 \pm 0.05	95.75	96.29 \pm 0.03
Task 5	95.91	95.96 \pm 0.05	91.65	90.84 \pm 0.01	94.21	93.15 \pm 0.08