On the Limitations of Continual Learning for Malware Classification

Mohammad Saidur Rahman, Matthew Wright
ESL Global Cybersecurity Institute
Rochester Institute of Technology
saidur.rahman@mail.rit.edu, matthew.wright@rit.edu

Scott E. Coull
Mandiant
scott.coull@mandiant.com

Abstract

Malicious software (malware) classification offers a unique challenge for continual learning (CL) regimes due to the volume of new samples received on a daily basis and the evolution of malware to exploit new vulnerabilities. On a typical day, antivirus vendors receive hundreds of thousands of unique pieces of software, both malicious and benign, and over the course of the lifetime of a malware classifier, more than a billion samples can easily accumulate. Given the scale of the problem, sequential training using continual learning techniques could provide substantial benefits in reducing training and storage overhead. To date, however, there has been no exploration of CL applied to malware classification tasks. In this paper, we study 11 CL techniques applied to three malware tasks covering common incremental learning scenarios, including task, class, and domain incremental learning (IL). Specifically, using two realistic, large-scale malware datasets, we evaluate the performance of the CL methods on both binary malware classification (Domain-IL) and multi-class malware family classification (Task-IL and Class-IL) tasks. To our surprise, continual learning methods significantly underperformed naive *Joint* replay of the training data in nearly all settings – in some cases reducing accuracy by more than 70 percentage points. A simple approach of selectively replaying 20% of the stored data achieves better performance, with 50% of the training time compared to *Joint* replay. Finally, we discuss potential reasons for the unexpectedly poor performance of the CL techniques, with the hope that it spurs further research on developing techniques that are more effective in the malware classification domain.

1 Introduction

Machine learning (ML) and deep learning (DL) models have become important tools in the defense of computers and networked systems. In particular, for addressing malicious software (*malware*), ML-based approaches have been used extensively in both academic literature and real-world systems for malware detection and classification, including for Windows malware (Tahan et al., 2012; Dahl et al., 2013; Johns, 2018), PDF malware (Maiorca et al., 2012; Laskov & Šrndić, 2011), malicious URLs (Lee & Kim, 2012; Stringhini et al., 2013), and Android malware (Arp et al., 2014; Grosse et al., 2017; Onwuzurike et al., 2019). These models are typically trained on previously observed samples and then deployed with the assumption that the model will generalize to new data. However, the adversarial nature of malware and continual evolution of benign software (*goodware*) makes for an inherently non-stationary problem.

To accommodate shifts in the data distribution over time, the model needs to be retrained regularly to maintain its effectiveness. Unfortunately, the speed with which new malware and goodware are produced results in large datasets that can be both costly to maintain and difficult to train on. For example, the AV-TEST institute registers more than 450,000 unique pieces of malware and "potentially unwanted applications (PUA)" each day (AV-TEST), while VirusTotal, a crowdsourced antivirus scanning service, regularly receives more than 1 million unique pieces of software each day (VirusTotal). Over the lifetime of a malware classification model, these daily feeds can result in datasets containing upwards of a billion unique training samples spanning multiple years. Given the realities of training these models, antivirus companies must decide whether to: (i) remove some older samples from the training set, at the risk of allowing attackers to revive older malware instead of writing new ones; (ii) train less frequently, at the cost of not adjusting to changes in the distribution; or (iii) expend tremendous effort to frequently retrain over all the data.

Continual learning (CL) offers an appealing alternative to these options by enabling incremental incorporation of new information and adaptation to data distribution shifts without maintaining large datasets or incurring significant training overhead. In this work, we investigate the extent to which malware classification models suffer from catastrophic forgetting (McCloskey & Cohen, 1989; Ratcliff, 1990; French, 1999), and whether we can address this using approaches from continual learning research. While combating the problem of catastrophic forgetting has been extensively studied, explored, and applied in the context of computer vision (Shin et al., 2017; Hsu et al., 2018; van de Ven & Tolias, 2018; van de Ven et al., 2020; Ji & Wilson, 2007; Farquhar & Gal, 2018) using the MNIST, CIFAR10 and CIFAR100, and ImageNet datasets, its role in and applicability to malware classification tasks remains unknown.

Using two large-scale malware datasets – Drebin (Arp et al., 2014) and EMBER (Anderson & Roth, 2018) – we examine the problems of binary malware classification and multi-class classification in the context of three CL scenarios: *domain incremental learning (Domain-IL)*, *class incremental learning (Class-IL)*, and *task incremental learning (Task-IL)*. For Domain-IL, we focus on the binary classification task of labeling software as malicious or benign, and consider the problem of incorporating shifts in the data distribution over time without losing the ability to classify older samples. For Class-IL and Task-IL, we examine the task of malware *family* classification, where a piece of malware is categorized into a well-defined family based on its code base, capabilities, and overall structure. In Class-IL, we incrementally add newly-discovered families at each iteration to mirror the ever-expanding universe of malware families found in the wild. The Task-IL formulation also incrementally adds new families to be classified, but constrains the classification task. All three scenarios represent problems faced in the anti-malware industry.

For each of these settings, we study 11 proposed CL approaches in our experiments, representing three major categories: regularization, replay, and replay with exemplars. We investigate their ability to reduce catastrophic forgetting in our two malware datasets compared with the baselines of (i) using no CL techniques and (ii) full *Joint replay*, which retrains on the full available dataset at each iteration. Finally, we investigate how much stored data may be enough to be replayed while still achieving high accuracy with lower storage and retraining costs. Our contributions are as follows:

- We are among the first to investigate continual learning in security problems, specifically in malware classification using deep learning models.
- We applied 11 distinct CL techniques in three CL scenarios using two large-scale malware datasets. We find that several CL techniques work reasonably well on Task-IL.
- We empirically show that none of the CL techniques are effective in the Domain-IL setting.
- For Class-IL, 10 of the 11 methods performed poorly, with only iCaRL (Rebuffi et al., 2017) performing marginally better against the *Joint* replay baseline.
- Instead of storing all prior data, we find that replaying 20-50% data is enough to significantly outperform all CL techniques in Domain-IL while also reducing the training cost by 35-50% compared with 100% *Joint* replay.
- We discuss the probable causes of the poor performance offered by CL techniques to help spur future research in this problem setting.

2 Related Work

Over the years, numerous mechanisms have been proposed to overcome catastrophic forgetting, which fall into three major families (Parisi et al., 2019): i) *regularization* methods, ii) *replay* methods, and iii) *adaptive expansion* methods.

Regularization-based techniques attempt to penalize changes to weights that are determined to be important to the previous tasks. This is done by introducing a new loss function known as *regularization loss*. The regularization loss is added to the training loss to compute the total loss. This category includes Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017), EWC Online (Schwarz et al., 2018), Synaptic Intelligence (SI) (Zenke et al., 2017), Memory Aware Synapses (Aljundi et al., 2018), Riemannian Walk (RWALK) (Chaudhry et al., 2018), Online Laplace Approximator (Ritter et al., 2018), Hard Attention to the Task (Serra et al., 2018), and Learning without Memorizing (Dhar et al., 2019).

Replay methods are designed to complement the training data for each new task with older data that are representative of the tasks seen so far (Chaudhry et al., 2019b; Rolnick et al., 2019). Distillation loss, for instance, is often used in replay-based techniques (Li & Hoiem, 2017; Rebuffi et al., 2017; Castro et al., 2018). Learning without Forgetting (LwF) (Li & Hoiem, 2017) replays *pseudo-data*, which is generated by first running the older model on new data to generate the softmax outputs, and then using those outputs as soft labels for training the model. This technique is similar to model distillation.

In an alternative form of replay called *generative replay (GR)* (van de Ven et al., 2020; van de Ven & Tolias, 2018; Shin et al., 2017), a second model is trained to capture the distribution of data from the previous tasks and generate new samples from the distribution to be replayed. In the current task T, both the main model $\mathcal{M}_T$ and a generative model $\mathcal{G}_T$ is trained using a mix of task T data and data generated by the prior generator model $\mathcal{G}_{T-1}$. GR is a powerful technique when access to older data is not available or restricted. Replay through Feedback (RtF) (van de Ven &

Tolias, 2018) reduces the training cost for dual-memory-based GR by coupling the generative model into the main model. RtF enhances the capability of the main model to generate samples by adding feedback connections and a layer of latent variables z, which are responsible for reconstructing inputs. Brain-Inspired Replay (BI-R) (van de Ven et al., 2020) is an improvement over RtF and proposes three new components: i) replacement of the standard normal prior with Gaussian mixture, ii) internal context (Masse et al., 2018), and iii) internal replay.

Another variant of replay, which intelligently picks the samples (i.e., called exemplars) to complement the training data of the current task, includes Experience Replay (Rolnick et al., 2019), Gradient Episodic Memory (GEM) Lopez-Paz & Ranzato (2017), Averaged Gradient Episodic Memory (A-GEM) Chaudhry et al. (2019a), and Incremental Classifier and Representation Learning (iCaRL) Rebuffi et al. (2017). ER leverages a reinforcement learning based learner to learn new and old experiences. In particular, ER balances stability and plasticity using on-policy for plasticity and off-policy for stability. iCaRL sets a memory budget beforehand, and the memory budget is equally divided into the previously learned classes. It picks and stores only exemplars that are close to the feature mean of each class. Loss of the current classes is minimized along with the distillation loss between targets obtained from the predictions of the previous model and the predictions of the current model on the previously learned classes. A-GEM also follows iCaRL to replay selective samples during training from the learned tasks to be replayed in the current task. When the original data is available, however, prior work has recommended *exact replay* instead (Rebuffi et al., 2017; Nguyen et al., 2017; Kemker & Kanan, 2017).

Adaptive expansion methods grow the capacity of the neural network as new tasks are observed. Several techniques have been proposed, including Progressive Neural Networks (Rusu et al., 2016), Dynamically Expandable Networks (Yoon et al., 2017), Adaptation by Distillation (Hou et al., 2018), and Dynamic Generative Memory (Ostapenko et al., 2019). These methods require incremental increases in memory as the new tasks are observed, and thus face scalability problems. Given the scale of malware classification problems, we leave the investigation of these techniques to future work, and instead focus on methods that do not require us to increase model capacity with dataset or problem size.

Finally, we note that no prior work has explored continual learning to the malware domain. However, Amalapuram et al. (2021) has proposed a continual learning-based network intrusion detection system (IDS) to mitigate catastrophic forgetting in a class-incremental scenario leveraging partial replay-based approaches. While they primarily focus on the role of class imbalance, some of the results of their study mirror our own findings, namely that the number and selection of samples used during replay are key to the success of continual learning in the cybersecurity space. We provide additional details and hypotheses for the underlying cause of this behavior given the unique nature of the malware classification problem in Section 6.

Additionally, there is some prior work on *online learning* (OL) applied to malware classification (Narayanan et al., 2016; Huynh et al., 2017; Narayanan et al., 2017; Xu et al., 2019). Online learning considers the problem of incorporating new samples into the model as they are observed, and notably does not directly address the problem of catastrophic forgetting. Furthermore, previous work has shown that producing high-quality labels for malware can be a difficult task (Kantchelian et al., 2015), and often requires weeks of time for vendors to come to consensus on newly-discovered variants (Zhu et al., 2020), making immediate incorporation of observed samples risky. Recently, researchers have explored malware detection systems leveraging transfer learning in an attempt to address the challenge of adapting to ever-evolving malware samples (Hsiao et al., 2019; Tang et al., 2020; Ale et al., 2020; Rong et al., 2021; Chai et al., 2022). Transfer learning, however, is not focused on retaining knowledge of the prior tasks when being applied to the new domain. In malware, this distinction is significant, since an inability to detect previous malware variants would reintroduce vulnerabilities without the user's knowledge. We thus focus in this paper only on CL techniques, as they explicitly address the issue of catastrophic forgetting.

3 Preliminaries

3.1 Training Protocols

Training in Continual learning (CL) involves considering three sets of parameters (Li & Hoiem, 2017) – i) θ_s: parameters that are shared across all the tasks, ii) θ_0: parameters specific to the previous tasks, and iii) θ_n: parameters specific to the new task. Here, θ_0 and θ_n contain the corresponding weights and the output layer of the older tasks and the new task, respectively. The weights of all the layers except the classification layer belong to θ_s.

Traditional deep learning (DL) training involves optimizing all three sets of parameters together, and is referred to as *Joint* training. This training method requires the availability – and hence the storage – of all the older data from previous tasks. The performance of this training mechanism is considered to be an upper bound, as data from all the previous tasks as well as the new tasks are used to train the model. Training this way, however, is slow and costly.

In CL training, for each new task n, θ_s and θ_n are optimized and updated while trying to keep θ_0 fixed. CL training does not require the older data to be available, which makes this a difficult task. Significant effort has been spent to boost the performance of models on the tasks in cases where there is no previous data available. Training is typically significantly faster than in *Joint* training, which may enable more frequent retraining of the model to keep up with changing data distributions or other needs without risking earlier concepts.

3.2 Continual Learning Scenarios for Malware Classification

We setup our continual learning problem in such a way that a ML model, M, learns a series of sequential tasks, $t_0, t_1, ..., t_n$. During the training of t_i, only data from t_i is available. Below, we describe three continual learning scenarios for malware classification (van de Ven & Tolias, 2019): Domain Incremental Learning (Domain-IL), Class Incremental Learning (Class-IL), and Task Incremental Learning (Task-IL).

Domain-IL. The most important problem in malware classification is binary classification of a test sample as either benign software (*goodware*) or malware. Many new malware samples, unidentified applications, and goodware are created and released every day. New malware and goodware often behave differently from prior ones, resulting in concept drift over time. This makes it valuable to incorporate new samples into production systems sooner than later. Previously proposed malware classification techniques generally do not consider an evolving model trained continuously to incorporate this distribution shift without complete retraining (Yang et al., 2021; Jordaney et al., 2017). Further, in this adversarial setting, an attacker may leverage older malware specifically to evade classifiers that have forgotten about it. Thus, the distribution shift must be captured while avoiding catastrophic forgetting.

In this binary malware classification setting, we divide our dataset into monthly tasks, where each month captures the natural concept drift of both malware and goodware due to evolving malware capabilities and benign software releases. We seek to incorporate this new knowledge in each monthly incremental learning iteration while maintaining previous discriminative knowledge about earlier malware and goodware.

We note that, unlike most studies of Domain-IL that rely on artificial manipulations of image datasets, the malware dataset we use shows real-world shifts in the data distributions over time. For example, we find that some malware families become more or less popular during the span of the dataset.

Class-IL. The second type of malware task is multi-class family classification. A *family* is a set of malware programs that have significant overlap in their code, such that they are considered by experts to be a group with common functionality. The famous Zeus banking trojan, for example, has evolved since 2006 to include 556 versions of software spread out among 35 different families with names like Citadel and Gameover (Schwarz, 2022).

It takes some time to label new malware samples and unknown applications with the help of experts and, in many cases, a consensus of scores from multiple anti-virus engines. Classes can be added when enough samples share enough similarity for the experts to decide that they are a new family (Kantchelian et al., 2015; Zhu et al., 2020). This occurs fairly infrequently in practice, but it requires the model to be adaptable to incorporate this new knowledge.

In this multi-class malware classification setting, we divide our dataset so that the model would learn new malware classes incrementally, extending its capabilities. We assume that the base model would start with a non-trivial number of classes, and then we would increment with new classes afterwards. Each new task is defined as adding new classes, but test time performance is measured on all classes that the model has been trained on so far.

In practice, an analysis pipeline is used to detect and triage malicious programs. The first step is binary classification of benign/malicious files. Once a file is determined to be malicious, it is useful to provide additional context to the security analyst about the malware family or its capabilities, which is encapsulated in Class IL. Since benign files are classified at the first stage in the pipeline, only malware is used in these settings.

Task-IL. To facilitate classification of malware into families, it can help to constrain the task based on information gained from other analysis methods, such as the broader category of the malware (e.g. adware, ransomware, etc.), behaviors of the malware (Berlin et al., 2015), or the infection vector that the malware uses (e.g. phishing, downloader, etc.). Task-IL captures this notion of constrained tasks, where adding a new task may represent a new category or new set of behaviors. This is likely to be less frequent than simply adding a new family, as we have in Class-IL, but it represents a real problem in malware classification. Unlike in Class-IL, the task identity is given to the model at test time, making it a much easier problem. In malware, this could mean learning the task identity from a separate model, manual analysis, or field reports of the malware's behavior. As we do not have naturally defined tasks in our datasets, we divide our dataset into tasks that contain an equal number of independent and non-overlapping classes, as is common in the continual learning literature (Kirkpatrick et al., 2017; Schwarz et al., 2018; Zenke et al., 2017).

3.3 Dataset

We used two popular, large-scale datasets from the malware research community for our experiments: Drebin (Arp et al., 2014) for Android malware and EMBER (Anderson & Roth, 2018) for Windows *Portable Executable (PE)* malware. We use EMBER 2018, as this is the latest version and is designed to be more difficult for the classifier.

Drebin. The Drebin dataset consists of 5,560 Android malware samples.[1] For our experiments, we use malware samples from the top 18 malware families totaling 4,525 malware samples. See Appendix A for more details.

Drebin features are organized as sets of strings, such as permissions, API calls, and network addresses, and they are embedded in a joint vector space as Boolean expressions representing the presence or absence of the attribute. Drebin features are highly sparse, making it a relatively easy dataset. The total number of original features in these samples is 8,803. With binary features, we do not need to use standardization, though we do pre-process the features using `scikit-learn's VarianceThreshold` (Pedregosa et al., 2011) to omit features with very low variance (< 0.001). This leaves a final set of 2,492 features.

We could only perform task-IL and class-IL experiments with this dataset. Even though the samples span over five years, the families in this dataset do not consistently contain enough samples for each time period.

EMBER. The EMBER 2018 dataset contains features from one million PE files scanned mostly in 2018, with 50K samples from before 2018.[2] There are 400K goodware samples, 400K malware samples, and 200K unknown samples.

EMBER contain a variety of features, including general file information, header information, imported and exported functions, and section information. EMBER features also contain three groups of format-agnostic features: byte histogram, byte-entropy histogram, and string information. Some of these features with high cardinality (e.g., identified strings) are then processed using the *hashing trick* (Weinberger et al., 2009) with different bin sizes. Each of the groups of EMBER features have unique distribution characteristics, which makes this dataset complex. It is also worth noting that the EMBER feature space encodes rich semantics for the underlying executable data that naturally constrain the feasible regions of that space. For more detail, refer to Anderson & Roth (2018).

There are 2,381 features in total, and we use `scikit-learn's StandardScaler` (Pedregosa et al., 2011) to standardize the feature space. StandardScaler provides a `partial_fit` method, which can be updated incrementally to standardize the dataset using each month representing a continuous flow of data.

For the Task-IL and Class-IL experiments, we only use the malware samples from 2018, which belong to 2,900 families. As we found that the majority of the families contain only a few samples, we filtered out the families containing fewer than 400 samples. This left us with 106 families, from which we select the top 100 malware families containing 337,035 samples for our experiments.

For the study of Domain-IL, we take both goodware and malware samples from 2018, removing the unknown samples (see Appendix A). This subset of the data spans 12 months, January to December. We focus on binary classification for this set of experiments.

4 Experimental Settings

4.1 Model Selection and Training

For the purposes of our experiments, we standardize our model as a multi-layer perceptron (MLP). Our MLP model has four fully-connected (FC) layers with $[1024, 512, 256, 128]$ hidden units, using dropout ($rate = 0.5$) and batch normalization in each layer. We use SGD as the optimizer, with learning rate = 0.01, momentum $= 0.9$, and weight decay $= 0.000001$. To reach these values, we performed selective hyperparameter tuning, covering the number of layers, the number of hidden units, activation functions, optimization function, and learning rate.

This model attains an AUC score of $= 0.99512$. As a baseline, we note that Anderson & Roth (2018) built a LightGBM model for binary classification, and report a state-of-the-art AUC score of 0.99642 on EMBER 2018.

[1] https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html
[2] https://github.com/elastic/ember

There are, however, a few data- and experiment-specific changes to the optimizer and learning rate that we finalized after conducting multiple sets of experiments in each setting. For experiments with the Drebin dataset, we observed that an Adam optimizer with learning rate 0.001 provided us slightly better performance than SGD, so we switch to it for Task-IL and Class-IL experiments on Drebin. For Task-IL and Class-IL with the top 100 classes of EMBER, we observe that SGD with learning rate 0.001 gave better accuracy than learning rate 0.01. Meanwhile, for the Domain-IL experiments with EMBER, SGD with learning rate 0.01 yielded the best performance.

For all scenarios and training protocols, the model is trained in a sequential manner, such that each of the N tasks $t_1, t_2, t_3, ..., t_N$ are independent, and their corresponding data distribution is also independent. The model only has access to the data of the current task. We use the standard multi-class cross-entropy loss for all the experiments except in Domain-IL. As Domain-IL is binary, we use binary cross entropy loss. We use mini-batch sizes of 32 and 256 for Drebin and EMBER, respectively.

4.2 Implementation Details

In Task-IL, we equally divide the number of classes into nine tasks for Drebin and and 20 tasks for EMBER data, where each task has two classes and five classes, respectively. In Class-IL with Drebin data, the model starts with 10 classes, and then we add two classes in each task, making five tasks in total. For EMBER, the model starts with 50 classes and then five classes are added in each task, making 11 tasks in total. In Class-IL, a task means an episode of learning new class(es). In Domain-IL, there are 12 tasks, one for each month's data of malware and goodware.

The output layer of each of the scenarios is implemented differently. The output layers of Task-IL and Class-IL have a distinct output unit for each class to be learned. The major difference of these two scenarios, however, is in the *active* output units. In Task-IL, only the output units of the classes in the current task are active. In Class-IL, however, all the output units of all the classes seen so far are active. In Domain-IL, all the output units – both of them in our binary classification task – are active, as only the data distribution is changing. The `softmax` function only considers these *active* units while assigning probabilities. We performed each set of experiments 10 times using different random parameter initialization, and a random buffer population strategy to choose samples to be replayed from the stored data.

We used `PyTorch` (Paszke et al., 2019) and ran our models on a `CentOS-7` machine with an Intel Xeon processor with 40 CPU cores, 128GB RAM, and four GeForce RTX 2080Ti GPU cards, each with 12GB GPU memory. The code and processed dataset of this paper are available at `https://github.com/msrocean/continual-learning-malware/`.

4.3 Baselines

We have two baselines in this work: i) *None* and ii) *Joint*. In the *None* baseline, the model is trained sequentially without any CL techniques. The performance of this method can be interpreted as an informal lower bound on our results, though it is not a true lower bound in theory or practice. In the *Joint* baseline, the accumulated data of all the tasks observed so far are used to train the model. The performance of this mechanism can be interpreted as an informal upper bound, though it too is not a theoretical upper bound. *Joint* replay requires storage and training effort proportional to all the data of the tasks observed so far. It is expensive, but it ensures high performance across the entire dataset up to the current iteration. The effectiveness of a particular technique to overcome catastrophic forgetting can then be seen as its ability to move the accuracy from being near to the lower bound to near the upper bound.

4.4 Continual Learning Techniques Studied

In this work, we apply 11 widely studied CL techniques belonging to three major categories: regularization, replay, and replay with exemplars. We provide a brief overview of each of the studied techniques in this section. The details of each of these techniques are given in Appendix B.

Regularization. Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) quantifies the importance of the weights in terms of their impact on the previous tasks' performance. As proposed, however, EWC is not scalable to a large number of tasks, as the regularization term grows with the number of tasks. EWC Online (Schwarz et al., 2018) is a modified version of EWC proposed to overcome this limitation. Synaptic Intelligence (SI) (Zenke et al., 2017) is similar to EWC Online, but it uses a different method to measure the importance of weights and a different regularization loss.

Replay. For each task, LwF trains with both the hard labels, to adapt to the new data distribution, and the soft labels, to retain aspects of the old model. Generative Replay (GR) (Shin et al., 2017), on the other hand, replays representative

Table 1: **Summary of the Experiments.** The average accuracy (Mean) and minimum accuracy (Min) from all the tasks in each experiment. Results in **Bold** indicate accuracy values closer to *Joint* performance than *None*. **EWC-O**: EWC Online, **GR-D**: GR + Distill. Error range is omitted for the results with less than 1.0 standard deviation .

Approach	**Method**	**Drebin**				**Ember**					
		Task-IL		**Class-IL**		**Task-IL**		**Class-IL**		**Domain-IL**	
		Mean	Min	Mean	Min	Mean	Min	Mean	Min	Mean	Min
Baselines	None	85.3	63.7±6	45.3	19.7	75.7	60±3.5	26.6	09.2	93.1	91.3
	Joint	**99.7**	99.3	**99.0**	97.5	**97.1**	95±3	**87.7**	85±2.5	**95.9**	93.2
Regul.	EWC	85.1	62±9	46.3	20±2	85.9	72±16	8.4	00.1	92.8	90.0
	EWC-O	83.9	64±7	47.1	20±2	78.8	57±31	9.0	00.2	93.1	91.5
	SI	90.7	78±7	45.3	20.0	73.6	58±4	27.3	09.5	93.0	91.1
Replay	LwF	**94.9**	88±4	27.8	5±1.5	**93.9**	91±9	11.9	00.7	93.2	91.7
	GR	**99.1**	98.1	55.1	26.2	80.8	70±6	26.9	09.3	93.2	91.6
	GR-D	**99.3**	98.5	55.1	26.1	82.9	73±3	27.0	09.0	93.2	91.7
	RtF	**99.4**	98.9	55.0	25.7	77.7	68±8.5	26.6	09.1	93.1	91.2
	BI-R	**95.9**	88±6	58.7	30±2.5	86.9	81±4	26.7	9.0	93.4	91.6
Replay + Exemplars	ER	**99.6**	99.1	55.2	26.7	**94.0**	91±1	28.0	09.4	75.9	65±4.5
	A-GEM	92.6	79±5.5	47.8	20±2	**90.4**	82±3	28.0	09.9	77.5	67.4
	iCaRL	-	-	**96.2**	95±1	-	-	**62.8**	46±2.5	-	-

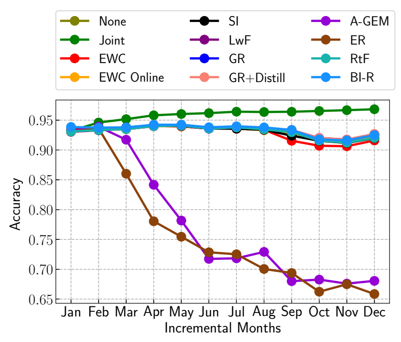


Figure 1: **Domain-IL on EMBER**: Accuracy over time.

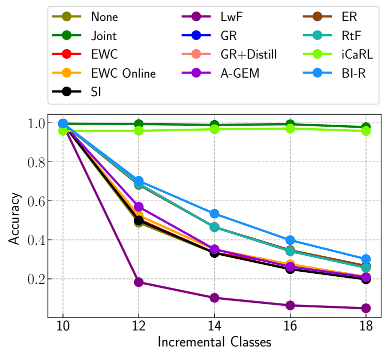


Figure 2: **Class-IL on Drebin**: Accuracy as the number of classes grows.

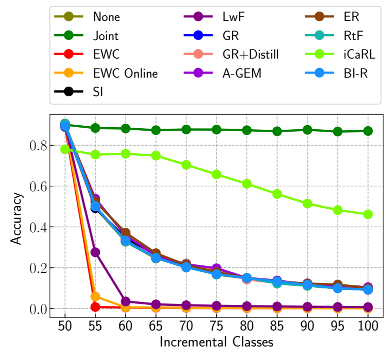


Figure 3: **Class-IL on EMBER**: Accuracy as the number of classes grows.

data of the previous tasks using a generative model. For the generative model, $\mathcal{G}$, in our experiments, we use a symmetric VAE (Kingma & Welling, 2013), in which the base model architecture is used for both the encoder and the decoder (van de Ven & Tolias, 2018; van de Ven et al., 2020). We use a variant of GR with distillation loss, as well. We also evaluate Replay through Feedback (RtF) (van de Ven & Tolias, 2018), which attempts to reduce the extensive cost of training the dual-memory-based GR technique by integrating the generative model into the main model. In addition, we study Brain-Inspired Replay (BI-R) (van de Ven et al., 2020) which improves upon RtF.

Replay + Exemplars. Experience replay (ER) (Rolnick et al., 2019) trains the model to learn new experiences (i.e., new tasks) coupled with replayed experiences (i.e., old tasks). iCaRL (Rebuffi et al., 2017) proposes to store $\mathcal{X}$ number of samples of the previously learned classes based on a memory budget. A-GEM (Chaudhry et al., 2019a) contains an episodic memory which stores a subset of the observed examples from previously learned task and replayed along with the new sample during training.

5 Evaluation

5.1 CL Experiments

In this section, we describe the results from our main set of experiments, investigating the three continual learning (CL) scenarios – Domain-IL, Class-IL, and Task-IL – using 11 continual learning techniques described in Section 4.4. We compare the results with two baselines – *None* and *Joint* – and perform experiments using two malware datasets – Drebin and EMBER. We summarize our findings in Table 1. The results of each of the experiments are represented in both **Mean** and **Min** metrics. Mean represents the mean accuracy of all the tasks in a single experiment, such as across 10, 12, 14, 16, and 18 classes tested for Class-IL with a given CL method on Drebin. Similarly, Min represents the minimum accuracy among all the tasks, which we highlight because the weakest performance shows the degree to which a technique may not be suitable for use.

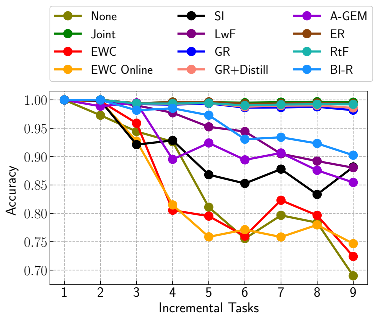


Figure 4: **Task-IL on Drebin**: Accuracy as number of tasks grows.

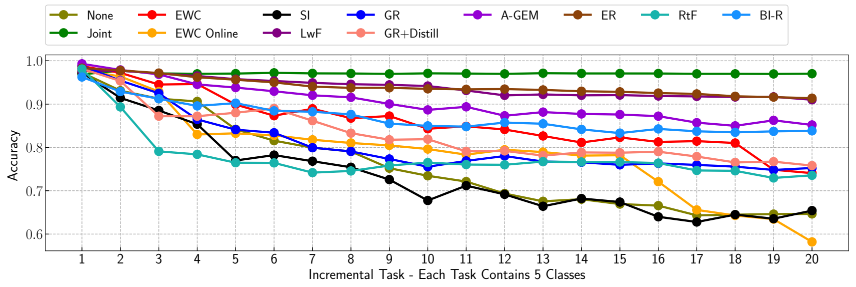


Figure 5: **Task-IL on EMBER**: Accuracy as the number of tasks grows.

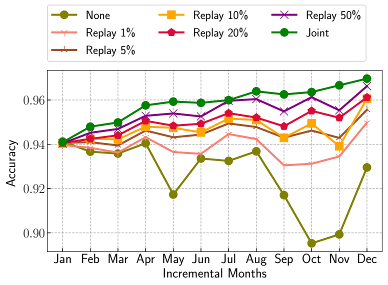


Figure 6: **Partial Joint Replay in EMBER**: Accuracy over time.

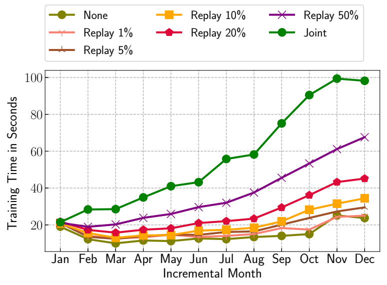


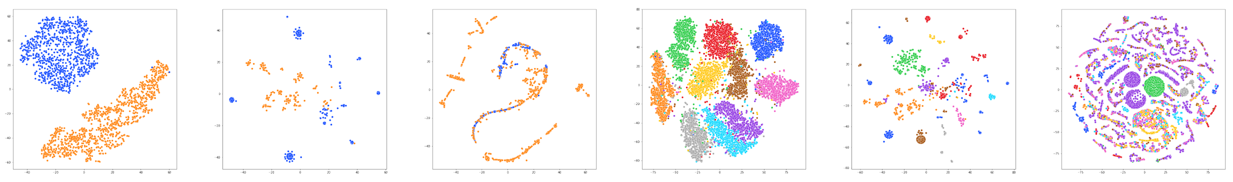
Figure 7: **Partial Joint Replay in EMBER**: Change in training time over time.

Domain-IL. In our experiments, we have 12 tasks representing the monthly data distribution shift of malware and goodware in EMBER from January to December 2018. Figure 1 shows the results, which are summarized in the rightmost column of Table 1. The mean accuracies of *None* and *Joint* baselines over the 12 tasks are 93.1% and 95.9%, respectively. We can see that Joint performance trends upward with each incremental task. This may be expected, despite changes in the data distribution, as there is more training data in each additional task. We also see that none of the CL techniques are effective, with all of them performing closer to *None* than to *Joint*. Note that the data distribution does not change dramatically during the year – even *None* reaches at least 91.3% or better in all cases.

Class-IL. Figure 2 and Figure 3 show the results of our experiments in this scenario for Drebin and EMBER, respectively. Since Class-IL is the most difficult CL scenario, it is not surprising that the mean accuracies for *None* and *Joint* are so far apart at 45.3% and 99.0%, respectively, on Drebin. All the regularization-based techniques and LwF perform poorly and very close to *None*. Among replay techniques, BI-R performs the best with 58.7% mean accuracy. iCaRL outperforms all the other CL techniques with 96.2% mean accuracy. On EMBER, the performance of all methods is about the same or even worse than *None* except iCaRL, which yields 62.8% mean accuracy. Even though iCaRL outperforms other techniques by a substantial margin, it still is far from the *Joint* baseline, which is notable given the volume of executables scanned by malware detection models each day – even a small increase in false positive or false negative rate can cause significant operational problems.

Task-IL. Figures 4 and 5 show the results of the Task-IL experiments on Drebin and EMBER, respectively. The average accuracies on the Drebin dataset of *None* and *Joint* training of the nine tasks, where each task contains two classes, are 85.3% and 99.7%, respectively. These form the effective lower and upper bounds in this setting. Among the regularization techniques, only SI yields closer to *Joint* level performance with 90.7% accuracy. The replay and replay-with-exemplars techniques are mostly effective, with GR, GR+Distill, RtF, and ER reaching over 99.0% average accuracy.

On EMBER, the average accuracy of *None* and *Joint* on the 20 tasks, where each task contains five classes, are 75.7% and 97.1%, respectively. None of the regularization-based techniques yield closer to *Joint* baseline performance. Among replay-based techniques, only LwF performs close to the *Joint* baseline with 93.9% mean accuracy. Replay-with-exemplars-based techniques – ER and A-GEM – outperform other techniques with 94.0% and 90.4% average accuracy, respectively. Considering both datasets, we can see that only ER performs reasonably well on both. GR, GR+Distill, RtF, BI-R, and SI all have difficulty with the larger and more complex EMBER dataset. In contrast, LwF does well on EMBER, but underwhelms on Drebin.



(a) MNIST 2-Class (b) Drebin 2-Class (c) EMBER 2-Class (d) MNIST 10-Class (e) Drebin 10-Class (f) EMBER 10-Class

Figure 8: Feature space visualization using t-SNE in Class-IL scenario.

5.2 Partial Replay with Stored Data

In this section, we perform a second set of experiments using *partial joint replay (PJR)*. This assumes that we have ample storage capacity and can maintain huge amounts of both historical and current data to be accessed at any time. Indeed, this may be the case for some companies doing malware detection. In this setting, CL techniques are not required, as the real data is available. Nevertheless, the huge volume of historical malware data makes it very expensive to train over all of it using full *Joint* replay. Thus, our question is how much of the historical data is needed to achieve high levels of accuracy using a strategy of sampled partial joint replay. In these experiments, we use early stopping with patience $= 5$, as replaying stored data causes the model to converge faster.

We focus on Domain-IL, which is the CL setting most applicable to malware classification. We performed seven sets of experiments with varying fractions – *None* (0%), 1%, 5%, 10%, 20%, 50%, and *Joint* (100%) – of the stored data to be replayed in the subsequent tasks. Figure 6 shows the results, while Figure 7 shows the corresponding training times. The mean accuracy of the *Joint* and *None* baselines is 96.4% and 93.0%, respectively.

The average accuracy over all the tasks with 1% of replayed data is 93.9% – nearly 1% higher than *None* and easily outperforming all CL techniques evaluated above. It is perhaps surprising to see that only 1% replayed data results in such a significant improvement in the average accuracy, but note that the total set of old training samples grows larger with each task, making it *multiple times larger than the new data* for most tasks in our experiment. Even a small sample of 1% thus becomes a significant fraction of all training data. When we increase the replayed data fraction to 5%, 10%, 20%, and 50%, average accuracy grows to 94.5%, 94.7%, 95.0%, and 95.4%, respectively. With 20% replayed data, the accuracy is only 1.4% lower than *Joint* replay. Reducing the replay data improves training efficiency significantly, as we can see in Figure 7. With our dataset, the expected amount of training effort using 20% of the replay data shrinks by 50% compared to full *Joint* replay. Even with 50% of the replay data, the expected training effort shrinks by 35%.

6 Discussion

Continual learning (CL) for malware classification can provide a number of benefits including: i) alleviating the need to store some or all of the previous data, ii) relaxing the requirement for access to previous data, iii) keeping training data private while sharing a model that can be updated, and iv) reducing computational cost. From our experiments, however, we can see that none of the CL techniques could contribute significantly in the Domain-IL setting, and only iCaRL provided reasonable performance in Class-IL setting, though even in that setting the gap between *Joint* and iCaRL was substantial – upwards of 24% on EMBER. For Task-IL, we observed reasonable performance by several methods on Drebin and on EMBER. Task-IL for malware, however, is of least significance, as adding new tasks is likely to occur less frequently than adding new classes or observing domain shift. Given these limitations, we now examine probable causes to help spur future research in CL considering these problem settings and datasets.

Dataset Complexity. Most papers on CL techniques use the MNIST dataset for their experiments, but MNIST is relatively simple in its data distribution and feature space. Figure 8 shows a t-SNE projection (Van der Maaten & Hinton, 2008) of the feature space to demonstrate the complexity of MNIST, Drebin, and EMBER dataset with both two and 10 classes for each. In MNIST, we can see a clear separation for the two classes and relatively clear separations for 10 classes. In Drebin, we see a somewhat sparser, yet complex space, particularly for 10 classes where some classes become difficult to separate. Meanwhile, the EMBER feature space is extremely complex with significant overlap between classes. Additionally, we see class imbalance in the EMBER dataset, which reflects the realistic sampling of real-world data. MNIST has equal numbers of samples in both Class-IL and Task-IL.

Another interesting distinction is the semantically-rich feature space of the malware classification datasets. Unlike the image domain, malware classifiers typically use tabular features derived from parsing and analysis of the binary program (e.g., headers, byte sequences, APIs). There are strong semantic constraints on the values that these features can take on and their relationships, which affects the shape of the feature space that feasible samples occupy. It is possible that these constraints and the inherent complexity of the classification task render generative, distillation, and regularization-based CL methods ineffective. Taken together, our results show that it is important to evaluate CL techniques on more complex, realistic datasets to understand how their effectiveness may generalize and apply in practical settings.

Real Domain Shifts. Prior work has studied Domain-IL only in a simulated experimental setting to mimic data distribution shift using the *permuted MNIST* protocol, in which pixels of the MNIST images are permuted around the image in a consistent way across the dataset to create the next task. This protocol clearly does not represent a realistic data distribution shift, and certainly not the strongly constrained feature space found in malware classification tasks. As such, the performance offered by CL techniques in this simulated setting is not a realistic measurement that would be likely to generalize to other data. In contrast, our experimental setting in Domain-IL represents a sample of real-world data that shifts over time. In Appendix C, we show a t-SNE visualization of MNIST and EMBER data distribution shift in Domain-IL. In summary, even though permuted MNIST is challenging for people to visualize, it appears to actually create reasonably solvable classification tasks for DL models. The natural distribution shift found in EMBER, on the other hand, creates very difficult tasks for the classifier.

Partial Joint Replay (PJR). Simple partial joint replay offers significant improvements over the *None* benchmark while also reducing computational effort. It would be interesting for future work to evaluate the effectiveness of combining partial joint replay and CL techniques, and our results here offer a pragmatic benchmark for future efforts. In some ways, the widely studied iCaRL CL technique is analogous to PJR in that iCaRL replays the samples stored in the memory buffer in the training phase, coupling those old samples with the new ones. As originally designed, however, iCaRL has a fixed memory budget that can limit its effectiveness with increasing classes. We can validate this by observing the performance of iCaRL for Drebin and EMBER datasets shown in Table 1 and Figures 2 and 3. Drebin has 18 classes and the performance gap between iCaRL and *Joint* is around 3%. On the other hand, EMBER has 100 classes and iCaRL's performance gap grows to almost 25%.

7 Conclusion

In this work, we investigate 11 continual learning techniques in three scenarios using two large, real-world malware datasets. Unfortunately, our findings demonstrate that in almost all cases those CL techniques are ineffective at preventing catastrophic forgetting and maintaining classification performance. Of the techniques evaluated, only iCaRL (Rebuffi et al., 2017) performed near *Joint* replay baselines. Meanwhile, we also found that a simple partial joint replay strategy of training on a small fraction of the historical data is enough to achieve reasonable performance with lower cost.

Taken as a whole, our results underscore the need for additional study of CL in complex, non-stationary problem settings. We hypothesize that the strong semantic constraints of the features used for malware classification tasks, along with the complex data distributions induced by natural drift in this space, lie at the heart of the performance differences between existing CL literature and the results in our paper. In particular, these unique aspects of the malware classification domain may severely limit the applicability of generative, distillation, and regularization-based methods due to their inability to sufficiently capture the inherent complexity. At the same time, these results also hint at possible avenues of future work. For example, the relative success of partial joint replay and iCaRL demonstrate the importance of sample selection during replay and the possibility of developing more effective strategies that ensure optimal coverage over feasible regions of the feature space. Most importantly, however, we hope this work spurs further exploration of CL in the cybersecurity domain and a broader conversation about real-world application of CL techniques.

Acknowledgements. We thank the anonymous reviewers for their helpful feedback. This material is based upon work supported in part by the National Science Foundation under Grant No. 1816851.

References

Laha Ale, Longzhuang Li, Dulal Kar, Ning Zhang, and Aayasha Palikhe. Few-shot learning to classify android malwares. In *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*, pp. 1001–1007. IEEE, 2020.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 139–154, 2018.

Suresh Kumar Amalapuram, Thushara Tippi Reddy, Sumohana S Channappayya, and Bheemarjuna Reddy Tamma. On handling class imbalance in continual learning based network intrusion detection systems. In *The First International Conference on AI-ML-Systems*, pp. 1–7, 2021.

Hyrum S Anderson and Phil Roth. EMBER: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*, volume 14, pp. 23–26, 2014.

AV-TEST. Malware statistics and trends report. https://www.av-test.org/en/statistics/malware/.

Daniel Bendor and Matthew A Wilson. Biasing the content of hippocampal replay during sleep. *Nature Neuroscience*, 15(10):1439–1444, 2012.

Konstantin Berlin, David Slater, and Joshua Saxe. Malicious behavior detection using windows audit logs. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, pp. 35–44, 2015.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 233–248, 2018.

Yuhan Chai, Lei Du, Jing Qiu, Lihua Yin, and Zhihong Tian. Dynamic prototype network based on sample adaptation for few-shot malware detection. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 532–547, 2018.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *International Conference on Machine Learning (ICML)*, 2019a.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. In *International Conference on Machine Learning (ICML)*, 2019b.

Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *International Conference on Acoustics, Speech, & Signal Processing (ICASSP)*, pp. 3422–3426. IEEE, 2013.

Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5138–5146, 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. In *International conference on machine learning (ICML)*, pp. 1407–1416. PMLR, 2018.

Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014.

Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security (ESORICS)*, pp. 62–79. Springer, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Lifelong learning via progressive distillation and retrospection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 437–452, 2018.

Shou-Ching Hsiao, Da-Yu Kao, Zi-Yuan Liu, and Raylin Tso. Malware image classification using one-shot learning with siamese networks. *Procedia Computer Science*, 159:1863–1871, 2019.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Ngoc Anh Huynh, Wee Keong Ng, and Kanishka Ariyapala. A new adaptive learning algorithm and its application to online malware detection. In *International Conference on Discovery Science (ICDS)*, pp. 18–32. Springer, 2017.

Daoyun Ji and Matthew A Wilson. Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature Neuroscience*, 10(1):100–107, 2007.

Jeff Johns. MalwareGuard: FireEye's Machine Learning Model to Detect and Prevent Malware. https://www.fireeye.com/blog/products-and-services/2018/07/malwareguard-fireeye-machine-learning-model-to-detect-and-prevent-malware.html, 2018.

Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *USENIX Security Symposium*, pp. 625–642, 2017.

Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, pp. 45–56, 2015.

Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

William DS Killgore, Ellen T Kahn-Greene, Erica L Lipizzi, Rachel A Newman, Gary H Kamimori, and Thomas J Balkin. Sleep deprivation reduces perceived emotional intelligence and constructive thinking skills. *Sleep Medicine*, 9(5):517–526, 2008.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Pavel Laskov and Nedim Šrndić. Static detection of malicious javascript-bearing pdf documents. In *Annual Computer Security Applications Conference (ACSAC)*, pp. 373–382, 2011.

S. Lee and J. Kim. WarningBird: Detecting Suspicious URLs in Twitter Stream. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(12):2935–2947, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

Davide Maiorca, Giorgio Giacinto, and Igino Corona. A pattern recognition system for malicious pdf files detection. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pp. 510–524. Springer, 2012.

Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. Adaptive and scalable android malware detection through online learning. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 2484–2491. IEEE, 2016.

Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, 2017.

Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.

Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11321–11329, 2019.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research (JMLR)*, 12:2825–2830, 2011.

Steve Ramirez, Xu Liu, Pei-Ann Lin, Junghyup Suh, Michele Pignatelli, Roger L Redondo, Tomás J Ryan, and Susumu Tonegawa. Creating a false memory in the hippocampus. *Science*, 341(6144):387–391, 2013.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285, 1990.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2001–2010, 2017.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3738–3748, 2018.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 350–360, 2019.

Candong Rong, Gaopeng Gou, Chengshang Hou, Zhen Li, Gang Xiong, and Li Guo. UMVD-FSL: Unseen Malware Variants Detection Using Few-Shot Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Dennis Schwarz. zeusmuseum. https://zeusmuseum.com/, 2022.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning (ICML)*, pp. 4528–4537, 2018.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in Neural Information Processing Systems (NeurIPS)*, 30:2990–2999, 2017.

Stelios M Smirnakis, Alyssa A Brewer, Michael C Schmid, Andreas S Tolias, Almut Schüz, Mark Augath, Werner Inhoffen, Brian A Wandell, and Nikos K Logothetis. Lack of long-term cortical reorganization after macaque retinal lesions. *Nature*, 435(7040):300–307, 2005.

Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *ACM Conference on Computer and Communications Security (CCS)*, pp. 133–144, 2013.

Gil Tahan, Lior Rokach, and Yuval Shahar. Mal-id: Automatic malware detection using common segment analysis and meta-features. *Journal of Machine Learning Research (JMLR)*, 13(1):949–979, 2012.

Zhijie Tang, Peng Wang, and Junfeng Wang. Convprotonet: Deep prototype induction towards better class representation for few-shot malware classification. *Applied Sciences*, 10(8):2847, 2020.

Gido M van de Ven and Andreas S Tolias. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018.

Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11(1):1–14, 2020.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9(11), 2008.

VirusTotal. VirusTotal - Stats. https://www.virustotal.com/gui/stats.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning (ICML)*, pp. 1113–1120, 2009.

Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. Droidevolver: Self-evolving android malware detection system. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 47–62. IEEE, 2019.

Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting and explaining concept drift samples for security applications. In *USENIX Security Symposium*, pp. 2327–2344, 2021.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *Journal of Machine learning research (JMLR)*, 70:3987, 2017.

Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *USENIX Security Symposium*, 2020.

A Dataset details

Drebin. We can see the details of Drebin dataset used in this study from Table 2. The total number of samples of the top 18 classes are 4525. Unfortunately, we could not find the samples belonging to *LinuxLotoor* and *GoldDream* in the dataset.

EMBER. Figure 9 shows the number of goodware and malware samples of each month of 2018.

Table 2: **Drebin.** 4525 malware samples from top 18 malware families.

Label	Family	#	Label	Family	#
0	FakeInstaller	925	9	Geinimi	92
1	DroidKungFu	667	10	Adrd	91
2	Plankton	625	11	DroidDream	81
3	Opfake	613	12	MobileTx	69
4	GingerMaster	339	13	FakeRun	61
5	BaseBridge	330	14	SendPay	59
6	Iconosys	152	15	Gappusin	58
7	Kmin	147	16	Imlog	43
8	FakeDoc	132	17	SMSreg	41
Total 4,525					

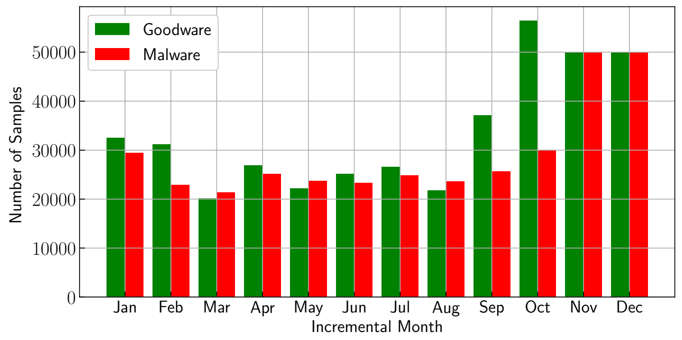


Figure 9: **EMBER Data**: Goodware and malware data statics of 12 months of 2018.

B Details of the Adapted Continual Learning Techniques

Elastic Weight Consolidation (EWC) and EWC Online. Kirkpatrick et al. (2017) proposed EWC to overcome catastrophic forgetting in a neural network. EWC is inspired by human brain, in which the plasticity of the synapses of the previously learned tasks are reduced to facilitate continual learning. As mentioned earlier, excessive plasticity of the weights of the previous tasks is the major cause of the catastrophic forgetting. If the plasticity of the weights is loosely connected to the previous task, then the network becomes less prone to catastrophic forgetting. Leveraging this idea, EWC quantifies the importance of the weights in terms of their impact on the previous tasks' performance, and selectively reduces the plasticity of the most important such weights.

A Bayesian approach is used to measure the importance of the parameters of a task in EWC. Given two tasks T_1 and T_2, EWC tries to converge to a point where both of these tasks have low error using the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_{T_2}(\theta) + \sum_i \frac{\lambda}{2} F_i \left(\theta_i - \theta^*_{T_1,i}\right)^2 \tag{1}$$

Here, $\mathcal{L}_{T_2}(\theta)$ is the loss of task T_2 only, $F_i(\theta_i - \theta^*_{T_1,i})^2$ approximates a Gaussian distribution with mean given by the parameters $\theta^*_{T_1}$ which is with respect to task T_1 and a diagonal of the Fisher information matrix F. i is an index into the weight vector. λ controls this distribution in such as way that the weights do not move too far away from the low error region of task T_1. Similarly, when a new task T_3 is observed, the loss function is updated in such a way that forces the parameters θ to be close to $\theta^*_{T_1,T_2}$, where $\theta^*_{T_1,T_2}$ is the parameters learned for the previous tasks T_1 and T_2.

The major drawback of the proposed EWC is scalability. The quadratic term of the regularization loss grows linearly with the increase of task which hinders to enable EWC be truly applicable in a practical continual learning scenario where the tasks keep growing. EWC Online attempts solve this problem by strict Bayesian treatment which results in a single quadratic penalty term considering the important parameters of the current task and a running sum of the Fisher Information matrices of the previous task's parameters (Schwarz et al., 2018). EWC Online modifies the second part of Equation 1 where $\tilde{F}_i$ is the running sum of the previous task. The modified loss function becomes as follows:

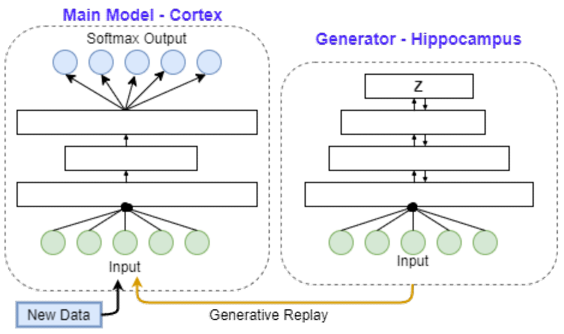


Figure 10: A schematic representation of Generative Replay.

$$\mathcal{L}(\theta) = \mathcal{L}_{T_2}(\theta) + \sum_i \lambda \tilde{F}_i \left(\theta_i - \theta^*_{T_1,i}\right)^2 \tag{2}$$

Synaptic Intelligence (SI) SI is a variant of EWC, in which changes to weights that are important to the previous tasks are penalized when training on new tasks (Zenke et al., 2017). Fundamentally, regularizer loss is added to the loss of the current task to get the total loss, $\mathcal{L}_{total}$. To compute this regularization loss, we first compute the importance of the weights I^w_t after every task t, with respect to the change in total loss $\mathcal{L}_{total}$. To get the estimated importance of the weights I^{N-1}_t for $N-1$ tasks, all the I^w_t are summed up together after a normalization step. Finally, the regularization loss $\mathcal{L}_r(\theta)$ for tasks $N > 1$ is computed in Equation 3. Refer to the paper (Zenke et al., 2017) for more details.

$$\mathcal{L}_r(\theta) = \sum_{t=1}^{K} I^{N-1}_t \left(\theta_t - \hat{\theta}^{(N-1)}_t\right)^2 \tag{3}$$

Experience Replay (ER) ER technique jointly trains the network utilizing both the examples (i.e., data) from the current task and examples stored in the very small episodic memory. ER is based on a distributed actor-critic training in which the single learner is fed both new and replayed experiences (i.e., tasks) (Rolnick et al., 2019). To adjust the off-policy distribution shifts during training, ER adapts V-Trace off-policy learning algorithm (Espeholt et al., 2018). Three losses – $L_{policy-gradient}$, L_{value}, and $L_{entropy}$ are common to both new and replayed experiences and another two losses – $L_{policy-cloning}$ and $L_{value-cloning}$ are unique to only the replayed experiences. Refer to the paper (Rolnick et al., 2019) for more details.

Learning without Forgetting (LwF) LwF tries to reduce the catastrophic forgetting of an older task t_{n-1} by learning the parameters of new task t_n, θ_n, with the help of the shared parameters θ_s and the parameters of the older tasks θ_0. The objective is to optimize θ_n and θ_s such that the prediction of t_n using θ_s and θ_0 does not drift significantly (Li & Hoiem, 2017). The objective function of LwF algorithm is given below:

$$\begin{aligned}\theta^*_s, \theta^*_0, \theta^*_n \longleftarrow \operatorname{argmin}_{\hat{\theta}_s,\hat{\theta}_0,\hat{\theta}_n} (\mathcal{L}_{new}(\hat{y}_n, y_n) + \\ \lambda_0 \mathcal{L}_{old}(\hat{y}_0, y_0) + \mathcal{R}(\theta_s, \theta_0, \theta_n))\end{aligned} \tag{4}$$

$\hat{y}_n$ is the prediction of the test samples of the new task t_n using current shared parameters $\hat{\theta}_s$ and current task's parameters $\hat{\theta}_n$. A multinomial logistic loss function is used to compute $\mathcal{L}_{new}$. Thus, $\mathcal{L}_{new}(\hat{y}_n, y_n)$ minimizes the difference between predicted $\hat{y}_n$ and actual y_n. y_0 is the prediction of the test sample of the new task t_n using the shared parameters θ_s and previous task's parameters θ_0 (i.e., the model before being trained with the samples of the new task). $\hat{y}_0$, on the other hand, is the prediction of the test samples of the new task t_n using current shared parameters $\hat{\theta}_s$ and previous task's parameters $\hat{\theta}_0$ (i.e., the model as trained with the samples of the new task). Distillation loss (Hinton et al., 2015) is used to compute the $\mathcal{L}_{new}$ so that the output of one network can be approximated using the outputs of another. Thus $\mathcal{L}_{old}(\hat{y}_0, y_0)$ minimizes the difference between $\hat{y}_0$ and y_0. λ_0 works as a balancing factor between the new task t_n and the previous task t_{n-1}. In the algorithm, $\mathcal{R}(\theta_s, \theta_0, \theta_n)$ works as a regularizer to avoid overfitting in the model.

Generative Replay (GR) and GR with Distillation Some of the earlier literature suggests that the cerebral cortex, which works as the main model, works better when paired with a generative model than a replay buffer (Killgore et al., 2008; Ramirez et al., 2013; Chen & Liu, 2018). In 2017, Shin et al. (Shin et al., 2017) proposed Deep Generative

Replay (DGR), in which the replayed examples of the previous tasks are generated using a Generative Adversarial Network (GAN) (Goodfellow et al., 2014), meaning that real data from the past tasks do not need to be stored. We can see a schematic representation of GR in Figure 10, where the generator works as the hippocampus and generates the representative samples of the stored data and replayed during the training of the model with new data. The GAN generator creates data to represent the model's knowledge of the previous tasks.

Given a series of tasks $t_1, t_2, ..., t_n$, an expert model $\mathcal{S}$ – which contains a generator model $\mathcal{G}$ and a solver model $\mathcal{M}$ (called the main model in some works (van de Ven & Tolias, 2018; van de Ven et al., 2020)) – holds the knowledge of the previous tasks and thus prevents the system from catastrophic forgetting. In this work, we adapt the DGR framework for our malware classification problem.

In DGR, $\mathcal{S}$ is learned and maintained in a continual learning fashion. $\mathcal{S}$ for the series of previous n tasks can be represented as $\mathcal{S}_n = (\mathcal{G}_n, \mathcal{M}_n)$. Given a new task t_{n+1} and the new training data D_{n+1}, the objective of $\mathcal{S}$ is to learn $\mathcal{S}_{n+1} = (\mathcal{G}_{n+1}, \mathcal{M}_{n+1})$. There are two steps involved in the learning process of $\mathcal{S}_{n+1}$ considering $D_{n+1} = (x, y)$ where (x, y) represents *(data, label)*:

1. At first, the *scholar* model $\mathcal{S}_{n+1}$ is updated with the input x of new task t_{n+1} and replayed with the generated data, $\hat{x}$, from previous scholar model $\mathcal{G}_n$. Real data x and replayed data $\hat{x}$ are mixed together at a ratio based on the importance of the new task t_{n+1} compared to the older task t_n. This is referred to as intrinsic replay or pseudo-rehearsal (Robins, 1995).
2. Then the main model $\mathcal{M}_{n+1}$) is trained with the real and replayed data with the following loss function:

$$\begin{aligned}\mathcal{L}_{train}(\theta_{n+1}) = r\,\mathbb{E}(x, y) \sim \mathcal{D}_{n+1}[\mathcal{L}(\mathcal{M}(x; \theta_{n+1}), y)] +\\ (1 - r)\,\mathbb{E}(x, y) \sim \mathcal{D}_n[\mathcal{L}(\mathcal{M}(x; \theta_{n+1}), y)]\end{aligned} \tag{5}$$

   Here, θ_n represents the parameters of the main model $\mathcal{M}_n$) and r represents the ratio of the mixture of the real and replayed data.

The DGR framework is designed in such a way that choice of the generative model is not limited to a GAN and can instead be a variational autoencoder (VAE) (Kingma & Welling, 2013) or any other such type.

For the experiments of GR, we need two models – the main model $\mathcal{M}$ and a generative model $\mathcal{G}$. $\mathcal{G}$ is responsible for generating representative samples of the previous tasks to be replayed in the current task. We use the base model architecture for both $\mathcal{M}$ and $\mathcal{G}$. The loss function of $\mathcal{M}$ consists of two parts – one for the data of the current task and another for the replayed samples. The cumulative loss of these two parts are weighted in terms of the number of tasks the model has observed so far.

For $\mathcal{G}$ in our experiments, we use a symmetric VAE (Kingma & Welling, 2013), where there is an encoder that maps the input data distribution to a latent distribution and a decoder that reconstructs the inputs from the latent distribution. For both the encoder and the decoder, the base model architecture is used. In all of our experiments, we used a stochastic latent variable layer with 100 Gaussian units parameterized by the mean and the standard deviation of the output of the encoder given input x.

The data to be replayed are sampled from the generative model, and then the selected samples are fed to the main model and then labeled based on the predicted class of the model. The samples to be replayed during task T are generated by the version of the main model and the generator after training on task $T - 1$. Hence, we need to store a copy of both $\mathcal{M}$ and $\mathcal{G}$ after each task.

GR with distillation is a variant of GR where the generated samples are replayed with the output probabilities (i.e., soft targets) instead of the actual labels. Previous work show that GR with distillation often works better than GR (van de Ven & Tolias, 2019; 2018; van de Ven et al., 2020).

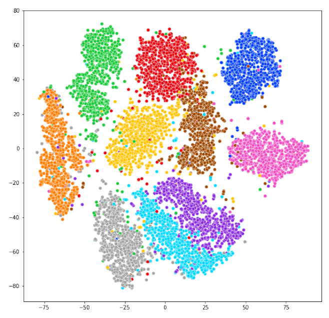
Replay through Feedback (RtF) GR has two models – a main model and a generative model. RtF proposes to merge the generator model into the main model (van de Ven & Tolias, 2018). This is inspired by the fact that replay in the brain is originated in the hippocampus and then it propagates to the cortex, and in our brain's processing hierarchy, the hippocampus sits on top of the cortex. The merged model will work as a brain, where the first n layers will work as the visual cortex and the last $m - n$ layers will work as the hippocampus. Technically, an additional `softmax` classification layer is added on top of the encoder of our generator VAE model. This technique requires only a single model to be trained, and the loss of the current tasks has two terms – cross entropy loss and generative loss.

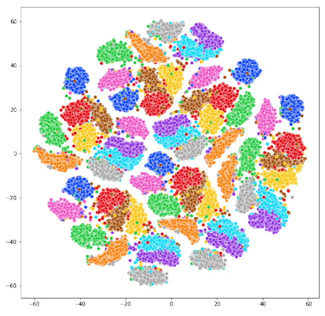
Brain-Inspired Replay (BI-R) Recently proposed by van de Ven et al. (2020), BI-R seeks to improve upon RtF with another three add-on components – Conditional Replay (CR), Gating based on Internal Context (Gating), and Internal Replay (IR). For CR, BI-R proposes to replace the standard normal prior over the VAE's latent variables by a Gaussian mixture with a separate mode for each class so that class-specific samples can be generated. This is due to the fact that a vanilla VAE is limited to generate class-specific samples, but humans do have control over which memories to recall. Conditional replay (CR) is intended to provide the network a human-like capacity to generate samples of the class the network needs most. Context-dependent gating was originally proposed by Masse et al. (2018). The idea is to reduce interference between different tasks by gating different and randomly selected network nodes for each task. However, this technique requires the task identity to be known for all the tasks, which is not realistic in the Class-IL scenario. BI-R proposes to use this gating technique in the decoder of the VAE with a conditional of the internal context. The task or class to be generated and reconstructed is the conditioned internal context. To note, not all of the nodes of the decoder network are gated. Mental images are not propagated all the way to the retina, and thus the brain does not replay memories to the input level (Bendor & Wilson, 2012). This insight is corroborated by evidence from neuroscience that our brain's early visual cortex does not change significantly from childhood to adulthood (Smirnakis et al., 2005). To accommodate these observations into continual learning, BI-R proposes to replay internally or at a hidden layer, instead of to the input level. From the machine learning perspective, the first n layers will need a limited amount of change, since there is no replay in them.

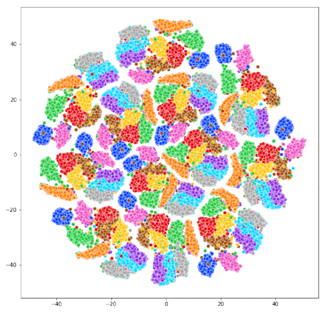
Incremental Classifier and Representation Learning (iCaRL) iCaRL (Rebuffi et al., 2017) is one of the earliest replay-based methods specifically designed for Class-IL scenario. Given a fixed buffer size (i.e., allocated memory), iCaRL stores samples of the earlier learned classes which are closest to the feature mean of those classes obtained from the feature maps of the network. iCaRL minimizes two loss functions – i) one is the categorical cross entropy loss of new classes, and ii) distillation loss obtained from the predictions of the current model's and the previous model's targets.

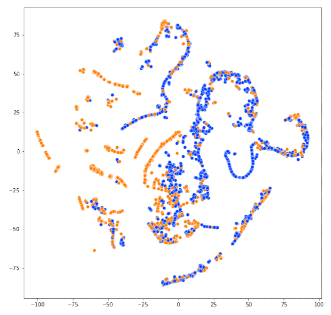
Averaged Gradient Episodic Memory (A-GEM) A-GEM (Chaudhry et al., 2019a) is an improved version of GEM(Lopez-Paz & Ranzato, 2017). GEM attempts to reduce catastrophic forgetting by constraining the updates of the new task not to interfere with the previous tasks. GEM utilizes the first order Taylor series approximation to estimate the direction of the gradient on the possible areas laid out by the gradients of the previously learned tasks. A-GEM relaxes the constraint to project the gradient into only one direction estimated from the randomly selected samples stored in a replay buffer. The replay buffer contains samples of the previously learned tasks.

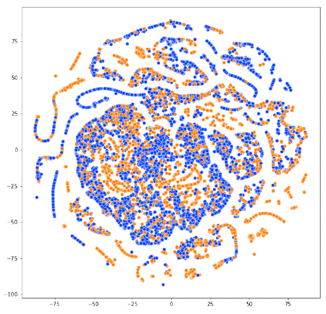
C FEATURE SPACE PROGRESSION IN DOMAIN-IL SCENARIO


(a) **Task 1**: Original MNIST (no permutation).


(b) **Task 6**: Cumulative MNIST data from Task 1 to Task 6 using *permuted* MNIST protocol.


(c) **Task 12**: Cumulative MNIST data from Task 1 to Task 12 using *permuted* MNIST protocol.


(d) **Task 1**: EMBER data of January


(e) **Task 6**: Accumulated EMBER data from January to June.

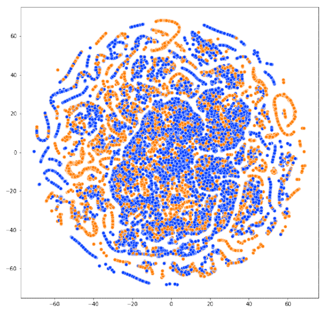

(f) **Task 12**: Accumulated EMBER data from January to December.

Figure 11: MNIST and EMBER data distribution shift in Domain-IL scenario using t-SNE plot.