On Handling Class Imbalance in Continual Learning based Network Intrusion Detection Systems

Suresh Kumar Amalapuram
Indian Institute of Technology Hyderabad
cs19resch11001@iith.ac.in

Thushara Tippi Reddy
Indian Institute of Technology Hyderabad
es17btech11021@iith.ac.in

Sumohana S. Channappayya
Indian Institute of Technology Hyderabad
sumohana@ee.iith.ac.in

Bheemarjuna Reddy Tamma
Indian Institute of Technology Hyderabad
tbr@iith.ac.in

Abstract

Modern-day cyber threats are growing more rapidly than ever before. To effectively defend against them, Anomaly-based Network intrusion detection systems (A-NIDS) must evolve continuously. Traditional machine learning techniques are ineffective in handling sequentially evolving tasks, and Neural Networks (NNs) in particular suffer from Catastrophic Forgetting (CF) of old tasks when trained on new ones. Continual Learning (CL) strategies help to mitigate CF by imposing constraints while training NNs on sequentially evolving data like network traffic. However, applying the CL framework in the design of A-NIDS is not straightforward due to the heavy Class Imbalance (CI) in the network traffic datasets. As a result, the performance of the system is very sensitive to the task execution order. In this work, we proposed a CL based A-NIDS by applying sample replay with Class Balancing Reservoir Sampling (CBRS) to mitigate CI in a Class Incremental Setting (CIS). Using the CICIDS-2017 dataset, experiments are conducted by permuting the majority class across the different task execution orders using the proposed CL based A-NIDS. We found that using auxiliary memory with context-aware sample replacing strategies, CF can be reduced to a greater extent, as opposed to data augmentation techniques which may alter the original data distribution and increase training time (with oversampling methods).

Keywords

Network intrusion detection system (NIDS), Continual learning, Deep learning, Catastrophic forgetting, Class imbalance.

1 Introduction

Intrusion Detection Systems (IDS) are an essential part of modern-day defense infrastructure to protect organizations from rapidly growing cyber threats. There is a necessity to incorporate sophisticated technologies in the design of these systems to defend against these modern-day threats. Deep Learning (DL) models have surpassed human performance in several computer vision tasks, and therefore system designers have gradually started incorporating it to build IDS. Yousefi-Azar *et al*. [40] used a single architecture of malware classification and anomaly detection where the features are learned using the AutoEncoder (AE). Singla and Bertino [33] provided a detailed analysis of various DL based methods in terms of their practicality for tasks like intrusion detection, malware analysis, and Botnet detection. Wang *et al*. [39] translated raw traffic into images and utilized Convolutional Neural Networks (CNNs) for representation learning to classify malware traffic. However, these DL based methods need to be self adaptable over time to cope with the ever-growing cyber attacks. Due to the non-stationary nature of threat data [1], the training process may become more complex, so the A-NIDS needs to self adoptable to the frequent data distribution changes. Continual Learning (CL) algorithms provide a promising direction to achieve the goals as mentioned above.

CL constitutes the ability to acquire, preserve, and fine tune the knowledge gained incrementally [6]. However, implementing CL using NNs on non-stationary data distributions results in the poor generalization of the model; this phenomenon is known as Catastrophic Forgetting (CF). Various approaches have been proposed to alleviate CF, that can broadly be categorized into three groups [29] as follows: 1) regularization based, 2) dynamic architecture, and 3) memory replay based. Regularization based methods mitigate CF by constraining the changes to the values of parameters of the model. Architectural properties are adjusted to prevent the interference of novel information with previously acquired knowledge in dynamic architecture based methods. Most appropriate information (samples) is replayed in memory replay based methods, and these replayed samples can either be from original train data or pseudo samples which are synthetically generated. As a consequence of employing these methods, CF can be reduced to a great extent which helps CL algorithms to achieve greater success in computer vision related tasks. However, CL algorithms may not be effective when applied to network data due to their diverse characteristics and are genetically different from image (vision) data. One of the most noticeable characteristics of the network data is *Class Imbalance* (CI), which poses a major challenge to the learning algorithm by introducing a bias towards the majority class, and thereby affecting the performance of A-NIDS [31].

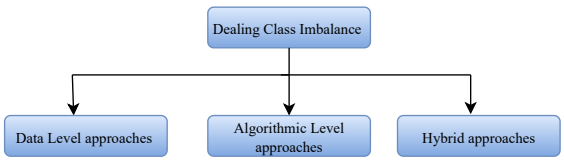


Figure 1: Approaches to deal with class imbalance data

CI is a critical problem seen in many real-world datasets (*e.g*., activity recognition, sentiment analysis, and text mining, etc.) [21]. Various techniques have been designed in the literature to address this issue, and they are broadly classified as data level, algorithmic level, and hybrid techniques (shown in Figure 1) [12]. Data

level techniques reduce the skewness in the data by resampling approaches, whereas algorithmic level techniques work by altering the objectives of the learning algorithms (cost sensitive learning). Hybrid approaches rely on both the balancing approaches mentioned above.

Johnson and Khoshgoftaar [18] presented various data, algorithmic level and hybrid approaches for class imbalance in the DNNs, and reported that nearly research was conducted with an emphasis on computer vision tasks using CNN. Delange *et al*. [8] discuss the task order effect (balanced, imbalanced) on an image dataset, and the related task orderings were created for the imbalance image dataset, and the notion of relatedness was captured using Expert Gate AE. In this work, we evaluate the impact of class balancing (using an algorithmic approach) on the effect of task execution order issue (see subsection 5.1) that arises due to the CI in the CL based A-NIDS in a CIS. However, our work differs from earlier works and is unique in the following ways: CL experiments are conducted on the network data which exhibits heavy CI compared to image data (where predominant CL literature was focused) , application of CL framework to intrusion detection is still in its nascent stage, and the performance results of our experimental setup which is build by permuting the majority class across the different task orders signifies the consideration of heavy CI in the design of new CL algorithms. Through this work we explore the efficacy of CL methods in the A-NIDS context while addressing the CI problem.

The remainder of this manuscript is organized as follows. The next section describes the related work on various class balancing approaches in the literature. Section 3 presents the proposed CL based A-NIDS architecture. Data processing details are discussed in Section 4 and performance results are present in Section 5. Finally, the conclusions and future directions are given in Section 6.

2 Literature Review

In this work, continual learning (also known as *life long learning* [36]) is applied to A-NIDS to make detection systems self-adaptable to the modern threats and also to reduce the re-training overhead. However, CL based learning algorithms are sensitive to the task execution order in which they are trained in due to heavy class imbalance in network traffic. On the contrary, class imbalance is relatively a rare issue in computer vision domain where CL is very effective. So, this section is devoted to reviewing various schemes that alleviate the class imbalance problem.

Data level (resampling) techniques, also known as preprocessing techniques applied before the training, include oversampling and undersampling. Oversampling techniques improve the minority class representation by generating additional samples; this generation process can be achieved in several ways. One of the simple ways for generation is to replicate the available minority class samples randomly (Random Oversampling (ROS) [2]), but this method may result in overfitting [23]. Another popular oversampling approach is to generate the synthetic samples (Synthetic Minority Oversampling Technique (SMOTE) [4]). All the oversampling techniques result in increased training time due to an increase in the training set size. In stark contrast to CI between different classes, the cluster based oversamlpling [17] technique will reduce the within-class imbalance. In recent times, Deep Neural Network (DNN) based generative methods gained popularity for generating synthetic samples. Variational AutoEncoder (VAE) [19], and Generative Adversarial Networks (GAN) [14], and conditional GAN [9] are familiar DNN based architectures for oversampling. Undersampling techniques will reduce the imbalance by eliminating the majority class representation, so these techniques will reduce train set size and time. One of the simplest techniques is Random undersampling (RUS) [34], which excludes the samples randomly, but these methods may remove the most informative samples [25]. However using informed undersampling techniques, this loss can be reduced [17, 27]. In general, undersampling methods are computationally effective and provide efficient storage against their counterpart oversampling. Hybrid approaches combine both oversampling and undersampling techniques [3].

Algorithmic level approaches mitigate the imbalance by fine-tuning the existing learners, for example, by varying the cost associated with each misclassification error (cost sensitive learning), adjusting the decision threshold (multi-class learning), and methods focusing one target class learning (one class learning) [20]. Cost sensitive learning is one of the popular algorithmic approaches, in which misclassification cost is heavily penalized to reduce the data skewness [22], but determining these penalty costs is a hard task (and depends on application domain) [13]. Wang *et al*. [38] introduce two novel loss functions to train DNNs with imbalanced data.

Hybrid approaches combine the techniques above to build robust learners [21], and ensemble based methods are more popular in this category. Tanha *et al*. [35] compare the significance of various boosting algorithms against 19 multi-class imbalanced datasets, so boosting techniques can be combined with resampling methods (*e.g*., *AdaC1, AdaC2, and Adac3*) [17] or cost sensitive approaches (*e.g*., *SMOTEBoost*) [5]. Ertekin [10] introduces an adaptive oversampling technique combined with active learning (VIRTUAL) and generates synthetic samples during the training process.

3 Proposed Continual Learning based A-NIDS System

This section briefly describes the proposed architecture that deals with class imbalance issues in a CL based A-NIDS. The proposed architecture (refer Figure 2) is designed with a goal to minimize the coupling between training and deployment processes, and therefore the architecture is functionally bisected into **Training** and **Deployment** segments. These two segments operate simultaneously in an asynchronous fashion, which helps to realize the continual learning of A-NIDS, in order to combat the ever-growing cyber threats. The training segment administers the data preparation activities and the training process of CL based learning algorithms. The deployment segment extracts the features from the incoming data stream and feeds them to the inference engine, which characterizes the nature (`Benign` or `Anomaly`) of the input data/traffic stream. In the following, we comprehensively explain each segment.

3.1 Training Segment

The success of any artificial learning system is severely affected by the quality of data used for training, especially in an A-NIDS setting where the lack of quality datasets is apparent due to privacy and

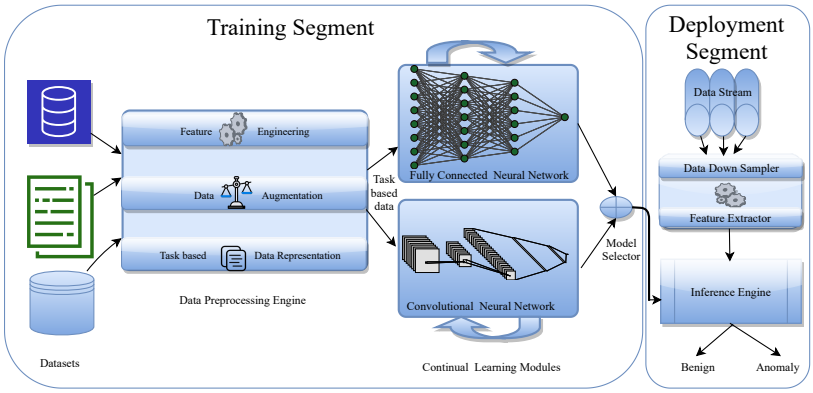


Figure 2: Architecture of the proposed CL based A-NIDS

security concerns. Due to the scarcity of these modern datasets (representatives of modern cyber threat data), it's an arduous task to add more intelligence to these detection systems to capture future cyber threats (zero-day attacks). One implicit way to overcome this issue is to employ CL based learning algorithms, which can be trained sequentially each and every time novel attack representative data is available. Additionally, CL algorithms also minimize the retraining overhead, requiring only limited data (specific to a particular task). The steady performance of CL algorithms will however be reduced by the CI Issue in the dataset. As remediation, we conduct experiments with various algorithmic level approaches to improve class balance and try to understand their effectiveness on performance metrics. So, enhancing the CL-based learning algorithm's performance using the class balancing approaches is the crux of the training segment. This entire workload is distributed between the **Data Preprocessing Engine** and **Continual Learning Module**, each of which are described in detail in the following

3.1.1 ***Data Preprocessing Engine:*** Data curation is an important functional aspect of any learning system, and performance degradation can be a consequence of not handling this well (due to missing values and noise in the data). The feature engineering peripheral is devoted to handling the aforementioned issues like the selection of most significant features and data massaging (fill missing values, data normalization, etc.). A more detailed discussion of these methods is presented in section 4. When employing data level approaches for dealing with CI, the **Data Augmentation** sub-module is active, which otherwise its an optional activity. Now, at this juncture data is relived from all the impurities and ready for the translation to a different representation known as **Task based** data representation. Each task essentially represents one or more class of attack data learn by the CL based algorithm, and all these different tasks are learned sequentially. In literature, this sequential learning is known as **Class Incremental Learning**.

3.1.2 ***Continual Learning Modules:*** CL module is a platform which acts as both container and executor for the CL based learning algorithms for A-NIDS. Constrained by the availability of hardware resources, various CL algorithms can be trained simultaneously, and better performers will be picked for deployment at the current moment, and this deployment process is repeated indefinitely. Additionally, this module is responsible for managing the algorithmic level approaches [7] when tackling CI, and we used the sample replay CBRS technique for this (see paragraph 3.1.2.1). Various experiments were conducted on two different model architectures. They are

Table 1: Model Architectures

(a) Simple MLP

Layer (type)	Output Shape	Param
Dense	[None, 100]	7100
Dropout	[None, 100]	0
Dense-1	[None, 1]	101

(b) Simple CNN

Layer (type)	Output Shape	Param
Conv2d	[None, 5, 8, 14]	140
Flatten	[None, 560]	0
Dense	[None, 100]	56100
Dropout	[None, 5, 8, 14]	0
Dense-1	[None, 1]	101

- **A Simple Multilayer Perceptron (MLP)**, containing single hidden layers with 100 neurons with Rectified Linear Unit (ReLU) activation. The output layer contains a single neuron with sigmoid activation.
- **A Simple Convolutional Neural Network (CNN)**, with one 2D convolutional layer having 14 output channels, uses 3×3 filters, and its output is flattened. A dense layer of 100 neurons with ReLU activations is followed. The penultimate layer is a dropout layer with a probability of `0.5`, and eventually, the output layer has a single neuron with sigmoid activation.

3.1.2.1 ***Sample Replay with CBRS:*** To understand the efficacy of algorithmic level class balancing approaches in the CL setting, we used sample replay with a memory population technique known as Class Balancing Reservoir Sampling (CBRS) [7]. In CBRS, we will have an auxillary memory (say size $\mathscr{M}$ units) used to store the ($\mathscr{M}$) informative training samples. The **Total Loss** ($\mathscr{L}$) function of CBRS has two components, one is for stream loss $\mathscr{L}_t$ for each task, other

loss $\mathscr{L}_m$ is for memory replayed samples, and these are balanced using a hyperparameter a (see Equation 1). The predictor function f_θ (parametrized by θ) is either a NN or a CNNs, which is trained using a CL setup. CBRS selects the most informative samples from the stream per each task, and stores them in the memory for sample replay.

$$\begin{aligned} \mathscr{L}_t &= \ell(f_\theta(x_t, y_t)) \\ \mathscr{L}_m &= \ell(f_\theta(x_m), y_m) \\ \mathscr{L} &= a \times \mathscr{L}_t + (1-a) \times \mathscr{L}_m \end{aligned} \tag{1}$$

We sketched two variants for balancing the samples in memory especially for the attack class. In **Whole Attack Class** Balance (WACB), all attack classes are considered as a single class and balanced along with benign samples, where as in **Individual Attack Class** Balance (IACB) each individual attack class is balanced.

3.2 Deployment Segment

Anomaly detection on streaming data brings additional challenges compared to its offline variant due to the non-stationary nature of the data (known as concept drift); as a result, the learning model becomes obsolete over time. To compensate for this effect, it is always desirable to deploy best-performing learning models that are trained on the most recent data. Dealing with high data flow rates of network traffic is the most neglected issue when discussing A-NIDS; employing a high speed cache is the most convenient solution but increases cost. Another straight forward approach is to use a **data down sampler**, which ignores a fraction of traffic data uniformly at random. **Feature Engineering** techniques are applied on the incoming stream to translate it into meaningful features which are fed to the **Inference Engine (IE)**. IE is the most visible peripheral/interface of the entire A-NIDS, and provides more flexibility to the end user in monitoring the network intrusion activities. IE is the placeholder for CL based learning model and provides an interface for easier deployment of the newer model over time. Eventually, IE classifies the incoming traffic data stream as `Benign` or `Anomaly`

4 Data Preparation

Learning the data distribution representation by training instances is always exciting, particularly when the training data inherently contains impurities (noise, missing values, etc.). This section is devoted to explaining how we processed the dataset to diminish these impurities.

4.1 Dataset Description

Canadian Institute for Cybersecurity (CIC) is the most renowned institute for generating various Network security datasets, and CICIDS 2017 is one such popular dataset. It is a labeled multi-attack class dataset, containing more than 80 network traffic flow based features extracted using CICFlowmeter. IDS2017 dataset contains a total of 15 class labels (`1 Benign` and `14 Attack` classes) [32]. Although this dataset is among the state of the art, it possesses several shortcomings. Some of them include heavy class imbalance, scattered presence of dataset (across different files), and huge volume of dataset [28]. The dataset was scattered across 8 different csv files (available in kaggle) that were used for our experiments [26]. All these 8 csv files are coalesced into a single large csv file before feature engineering.

4.2 Feature Engineering

CICIDS2017 dataset features which contains the values `NULL` or `Infinity` are replaced with `0`. Several redundant features are also found, which have their only value as `0`, and such features are removed. The removed list of features is Bwd PSH Flags, Bwd URG Flags, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, and Bwd Avg Bulk Rate. class *sklearn.preprocessing.LabelEncoder* was used to encode all the categorical features [30]. Our focus in this work is the design of A-NIDS, so all the attack class labels are replaced with the label `1`, and benign class labels with the label `0`. Prior to training, all the dataset values are normalised using *Min-Max* normalization technique (2), where χ_{ij} is the specific value of feature χ_j, and δ (non-zero value) was added to the denominator. Eventually, we have 70 features for training CL based A-NIDS .

$$x_{ij} = \frac{\chi_{ij} - \min(\chi_j)}{\max(\chi_j) - \min(\chi_j) + \delta} \tag{2}$$

4.3 Data Augmentation

Learning from imbalanced data has received massive attention from a larger audience due to the CI data in many real-world applications. As a result of CI, learning algorithms will be skewed towards the majority class; thus, minority classes are mostly neglected or ignored. The job of the data augmentation module is to lower the class imbalance utilizing the **data level/data augmentation** approaches. Some well-known data augmentation techniques are RO, Random Undersampling Random Oversampling (RURO), and variants of SMOTE. More than 80 variants of SMOTE are available, among which, some of the most familiar extension methods [11] are Regular SMOTE, Adaptive Synthetic sampling (ADASYN) [16], Borderline-SMOTE (BSMOTE) [15]. These techniques are applied during the preprocessing stage (as supplementary work), and sometimes may cause extra burden to the training process. For instance, when applied RURO on the CICIDS2017 dataset, the strength of minority class (Heartbleed, Web Attack Sql Injection, Infiltration etc.) samples are increased (see Table 2); however, these newer samples are replications of existing instances, thus learning algorithm may overfit the minority classes.

Table 2: Increased Minority Class Samples after RURO

	Web Attack XSS	**Infiltration**	**Web Attack Sql Injection**	**Heartbleed**
Number of samples in CICIDS2017	456	25	14	7
Number of samples after RURO	111251	111251	111251	111251

Another most noticeable difficulty with oversampling techniques is increased training set size that incur a heavy toll on training time. For example, see Table 3 which contains the training set sizes of the CICIDS2017 dataset after applying various data augmentation techniques. The original column indicates the training set size of CICIDS2017 (without data augmentation).

Table 3: Training Set Size after Oversampling

	original	**RO**	**RURO**	**SMOTE**	**BSMOTE**	**ADASYN**
Number of samples (in millions)	19.8	31.9	16.6	31.9	30.8	31.9

On the other hand, while undersampling techniques eliminates the majority class samples, one potential problem with this method could be the loss of most informative samples, as a result, original data distribution will be altered. However, informed search strategies can be used to preserve informative samples as opposed to RUS, but this may raise the complexity of the training process. Thus, motivated by these remarks on the data augmentation strategies, we shift our focus to the algorithmic ways to deal with CI, which essentially preserve the original data distribution.

4.4 Task based Data Representation

CL algorithms mimic human-like learning where the training data is seen as a sequence of tasks, and learn them incrementally [24]. Generally, each task's data is locally independent and identically distributed (*iid*), but in our case, each task will represent more than one attack class distribution to learn. The CICIDS2017 dataset is transformed into 5 tasks and tested under the **Class Incremental** setting [37].

4.5 Execution Environment

Experiments are conducted using Google Colab (pro) platform (with 25/35 GB memory, and Python 3 Google compute engine backend GPU/TPU), and a workstation (64-bit, 32 GB RAM, 2 GB GPU, and 6-Core processor). Keras software library is used for training the NNs, and CNNs.

5 Performance Results

We conducted the experiments to empirically understand the impact of class balancing approaches (algorithmic level) on the task order effect seen in the CL-Based A-NIDS. Towards this, we used one algorithmic level approach (CBRS [7]) with different buffer sizes using two NN (simple MLP and simple CNN) architectures. We used performance metrics in subsection 5.1 as our baseline (see Table 4). Due to the heavy CI in the dataset, considering `accuracy` metric alone will not be sufficient for a complete understanding of the various algorithms, so additional metrics `recall`, `Precision`, `F1-score` are used. All these metrics populate the **confusion matrix** (where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives)

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}, \tag{3}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}, \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}, \tag{5}$$

$$F1 = \frac{2 * \text{Precision*Recall}}{\text{Precision+Recall}}. \tag{6}$$

5.1 Effects of Task Execution Order

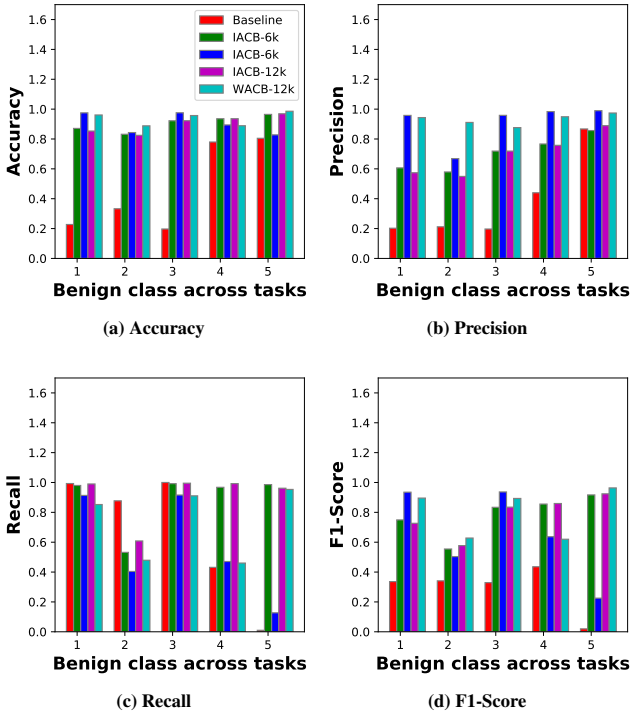
The CICIDS2017 training data was partitioned into 5 tasks, with each task containing three classes of the dataset. Five task orders were created by placing `Beningn` in each of this 5 tasks respectively. Specifically, the Benign class was placed only in one of the five tasks in each task ordering. The aim of this setting is to introduce heavy imbalance in the different task orders. It was noticed that different task execution orders result different performance as quantified by the metrics. This is due to permuting the Benign class (Major class) across the different task orders (see Table 4 and Table 5). This behavior is known as **Task Order Sensitivity**. These performance metrics are obtained by training a simple MLP (simple CNN) in a CIS on the original CICIDS2017 dataset. These results are used as a **baseline** for our remaining experiments.

Table 4: Task Execution Order Effect on Performance Metrics using CICIDS2017 dataset with Simple MLP Architecture

Task order	Accuracy	Precision	Recall	F1 Score
0 in task 1	0.227304	0.202313	0.993013	0.336141
0 in task 2	0.332665	0.211785	0.877167	0.341191
0 in task 3	0.197140	0.197029	1.000000	0.329197
0 in task 4	0.779760	0.439877	0.431500	0.435648
0 in task 5	0.804652	0.867224	0.009916	0.019608

Table 5: Task execution order effect on performance metrics using CICIDS2017 dataset with simple CNN architecture

Task order	Accuracy	Precision	Recall	F1 Score
0 in task 1	0.197784	0.197156	1.000000	0.329374
0 in task 2	0.187112	0.188565	0.946438	0.314474
0 in task 3	0.186216	0.185445	0.922887	0.308833
0 in task 4	0.250855	0.198235	0.920580	0.326222
0 in task 5	0.805242	0.933576	0.012265	0.024213



(a) Accuracy (b) Precision

(c) Recall (d) F1-Score

Figure 3: Comparison of performance metrics of the variants of CBRS using simple MLP architecture on the CICIDS17 dataset

5.2 Impact of Algorithmic Level Method

Sample replay with CBRS was used to understand the impact of the algorithmic level approach on task order sensitivity. We scrutinized this method extensively by varying some of its parameters like the memory size, and the way classes are loaded into the memory (WACB, IACB) using simple MLP, and simple CNN architectures. This is done by fixing the hyperparameter a value (to 0.3). A total of 8 different experiments were conducted (see Table 6). All these are partitioned into two groups based on the underlying architecture (see Figure 3 and Figure 4). For the first group, Figure 3 shows the accuracy, precision, recall, and f1-score on CICIDS dataset for experimental configuration using simple MLP architectures. The experiment with the WACB configuration, a memory size of 12000 units has optimal performance. All the individual performance metrics for this configuration are shown in Table 8, and the baseline metrics can be seen Table 4. The seconds group of experiments are carried out using the simple CNN architecture (see Figure 4), with the IACB configuration. A memory size of 6000 units yields the optimal performance, and more fine grained performance details of this configuration are available in Table 7, and the baseline metrics can be seen in Table 5.

Table 6: Various experimental strategies

S.No	Architecture	Sample Loading strategy	Memory Size (units)	Legend label
1	Simple MLP	IACB	6000	IACB-6k
2	Simple MLP	IACB	12000	IACB-12k
3	Simple MLP	WACB	6000	WACB-6k
4	Simple MLP	WACB	12000	WACB-12k
5	Simple CNN	IACB	6000	IACB-6k
6	Simple CNN	IACB	12000	IACB-12k
7	Simple CNN	WACB	6000	WACB-6k
8	Simple CNN	WACB	12000	WACB-12k

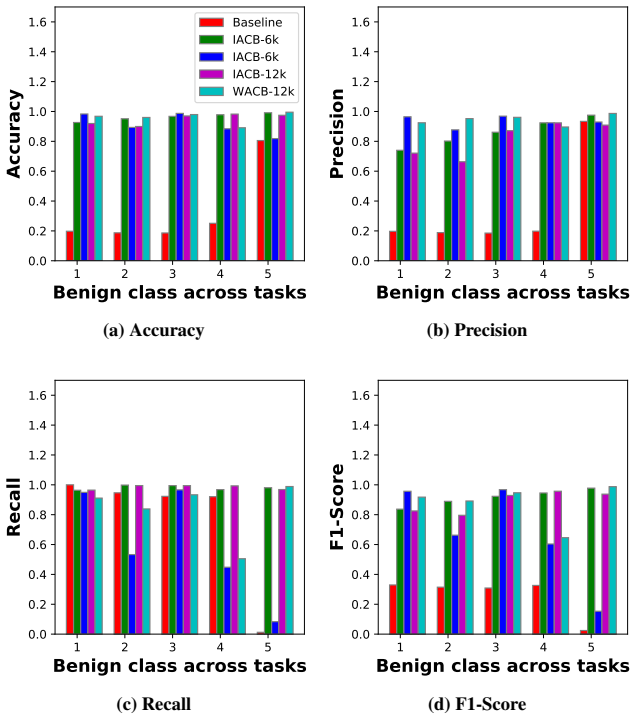
Table 7: Performance metrics of different task order using Sample Replay with CBRS (IACB, Memory=6000 units) with simple CNN architecture

Task order	Accuracy	Precision	Recall	F1 Score
0 in task 1	0.926151	0.740025	0.963682	0.837173
0 in task 2	0.951086	0.801905	0.998326	0.889400
0 in task 3	0.967334	0.860985	0.994806	0.923071
0 in task 4	0.977965	0.923883	0.967890	0.945375
0 in task 5	0.991245	0.975164	0.980532	0.977840

Table 8: Performance Metrics of different Task order using Sample Replay with CBRS (WACB, Memory=12000 units) with Simple MLP Architecture

Task order	Accuracy	Precision	Recall	F1 Score
0 in task 1	0.960857	0.943461	0.852391	0.895617
0 in task 2	0.888148	0.910910	0.479085	0.627921
0 in task 3	0.957066	0.876340	0.910550	0.893117
0 in task 4	0.888798	0.948913	0.460311	0.619909
0 in task 5	0.985835	0.974084	0.953467	0.9636650

This experimental study reveals that the task order sensitivity can be scaled down using the sample replay approach with appropriate memory population techniques. Another noticeable insight is that, increasing the auxiliary memory size may not yield the optimal results, and sometimes it depends on the underlying architecture also.



(a) Accuracy (b) Precision

(c) Recall (d) F1-Score

Figure 4: Comparison of performance metrics of the variants of CBRS using simple CNN architecture on the CICIDS17 dataset

6 Conclusions & Future Directions

It is essential to incorporate modern techniques into the design of A-NIDS to protect against rapidly proliferating cyber threats. The CL framework is one of the most promising approaches that provide more flexibility to evolve quickly. However, integrating CL algorithms to IDS is not straightforward due to the inherent impact of task execution order on the performance metrics due to the heavy CI. We explored algorithmic level methods to fix the CI problem that araises in the context of network data and CL framework. Algorithmic level methods work by altering the existing learner, and experiments are carried using sample replay with the CBRS algorithm. This method is shown to achieve promising results; additionally, we also observed the impact of buffer size on the performance metrics, which needs careful attention.

As future work, we plan to develop CL algorithms that identify the unseen classes during testing on the fly and incorporate them easily once labeled data is available (Open World Learning). We plan to identify challenges involved in bringing the notion of relatedness among the various tasks for creating more natural task execution ordering (curriculum learning). This work also brings into the limelight the importance of dealing with CI in general CL frameworks. Another interesting direction would be generating synthetic samples during the training process for the minority class, rather than generating at the preprocessing stage (data level approaches) [10].

References

[1] Blake Anderson and David McGrew. 2017. Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 1723–1732. https://doi.org/10.1145/3097983.3098163

[2] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explor. Newsl.* 6, 1 (June 2004), 20–29. https://doi.org/10.1145/1007730.1007735

[3] Silvia Cateni, Valentina Colla, and Marco Vannucci. 2014. A method for resampling imbalanced datasets in binary classification tasks for real-world problems. *Neurocomputing* 135 (2014), 32–41. https://doi.org/10.1016/j.neucom.2013.05.059

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority over-Sampling Technique. *J. Artif. Int. Res.* 16, 1 (jun 2002), 321–357.

[5] Nitesh V. Chawla, Aleksandar Lazarevic, Lawrence O. Hall, and Kevin W. Bowyer. 2003. SMOTEBoost: Improving Prediction of the Minority Class in Boosting. In *Knowledge Discovery in Databases: PKDD 2003*, Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 107–119.

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

[7] Aristotelis Chrysakis and Marie-Francine Moens. 2020. Online Continual Learning from Imbalanced Data. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 1952–1961. http://proceedings.mlr.press/v119/chrysakis20a.html

[8] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), 1–1. https://doi.org/10.1109/TPAMI.2021.3057446

[9] Georgios Douzas and Fernando Bacao. 2018. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with Applications* 91 (2018), 464–471. https://doi.org/10.1016/j.eswa.2017.09.030

[10] Şeyda Ertekin. 2013. Adaptive Oversampling for Imbalanced Data Classification. In *Information Sciences and Systems 2013*, Erol Gelenbe and Ricardo Lent (Eds.). Springer International Publishing, Cham, 261–269.

[11] Alberto Fernández, Salvador García, Francisco Herrera, and Nitesh V. Chawla. 2018. SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-Year Anniversary. *J. Artif. Int. Res.* 61, 1 (jan 2018), 863–905.

[12] Vaishali Ganganwar. 2012. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering* 2, 4 (2012), 42–47.

[13] Adel Ghazikhani, Reza Monsefi, and Hadi Sadoghi Yazdi. 2013. Online cost-sensitive neural network classifiers for non-stationary and imbalanced data streams. *Neural computing and applications* 23, 5 (2013), 1283–1295.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. (2014), 2672–2680.

[15] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In *Advances in Intelligent Computing*, De-Shuang Huang, Xiao-Ping Zhang, and Guang-Bin Huang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 878–887.

[16] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*.

[17] Haibo He and Edwardo A Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21, 9 (2009), 1263–1284.

[18] Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 1 (2019), 1–54.

[19] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. arXiv:1312.6114 [stat.ML]

[20] Sotiris Kotsiantis, D. Kanellopoulos, and P. Pintelas. 2005. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering* 30 (11 2005), 25–36.

[21] Bartosz Krawczyk. 2016. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence* 5, 4 (2016), 221–232.

[22] Charles Ling and Victor Sheng. 2010. Cost-Sensitive Learning and the Class Imbalance Problem. *Encyclopedia of Machine Learning* (01 2010).

[23] Alexander Liu, Joydeep Ghosh, and Cheryl Martin. 2007. Generative Oversampling for Mining Imbalanced Datasets. *International Conference on Data Mining*, 66–72.

[24] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient Episodic Memory for Continual Learning. (2017), 6470–6479.

[25] Octavio Loyola-González, José Fco. Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa, and Milton García-Borroto. 2016. Study of the impact of resampling methods for contrast pattern based classifiers in imbalanced databases. *Neurocomputing* 175 (2016), 935–947. https://doi.org/10.1016/j.neucom.2015.04.120

[26] Arash Habibi Lashkari man Sharafaldin and Ali A. Ghorbani. 2018. *CICIDS2017 Intrusion Detection Evaluation Dataset*. Retrieved July 2, 2021 from /https://www.kaggle.com/cicdataset/cicids2017/

[27] Inderjeet Mani and I Zhang. 2003. kNN approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, Vol. 126. ICML United States.

[28] Ranjit Panigrahi and Samarjeet Borah. 2018. A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering Technology* 7, 3.24 (2018), 479–482. https://doi.org/10.14419/ijet.v7i3.24.22797

[29] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (2019), 54–71. https://doi.org/10.1016/j.neunet.2019.01.012

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

[31] Sireesha Rodda and Uma Shankar Rao Erothi. 2016. Class imbalance problem in the network intrusion detection systems. In *International conference on electrical, electronics, and optimization techniques (ICEEOT)*. 2685–2688. https://doi.org/10.1109/ICEEOT.2016.7755181

[32] Iman Sharafaldin., Arash Habibi Lashkari., and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*. INSTICC, SciTePress, 108–116. https://doi.org/10.5220/0006639801080116

[33] Ankush Singla and Elisa Bertino. 2019. How Deep Learning Is Making Information Security More Intelligent. *IEEE Security and Privacy* 17, 3 (2019), 56–65.

[34] Muhammad Atif Tahir, Josef Kittler, Krystian Mikolajczyk, and Fei Yan. 2009. A Multiple Expert Approach to the Class Imbalance Problem Using Inverse Random under Sampling. In *Multiple Classifier Systems*, Jón Atli Benediktsson, Josef Kittler, and Fabio Roli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 82–91.

[35] Jafar Tanha, Yousef Abdi, Negin Samadi, Nazila Razzaghi, and Mohammad Asadpour. 2020. Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data* 7, 1 (2020), 1–47. https://doi.org/10.1186/s40537-020-00349-y

[36] S. Thrun. 1994. A lifelong learning perspective for mobile robot control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, Vol. 1. 23–30 vol.1. https://doi.org/10.1109/IROS.1994.407413

[37] Gido M. van de Ven and A. Tolias. 2019. Three scenarios for continual learning. *ArXiv* abs/1904.07734 (2019).

[38] Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J. Kennedy. 2016. Training deep neural networks on imbalanced data sets. In *International Joint Conference on Neural Networks (IJCNN)*. 4368–4374. https://doi.org/10.1109/IJCNN.2016.7727770

[39] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. 2017. Malware traffic classification using convolutional neural network for representation learning. In *International Conference on Information Networking (ICOIN)*. IEEE, 712–717. https://doi.org/10.1109/ICOIN.2017.7899588

[40] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. 2017. Autoencoder-based feature learning for cyber security applications. In *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK, USA). IEEE, 3854–3861. https://doi.org/10.1109/IJCNN.2017.7966342