This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

AESMOTE: Adversarial Reinforcement Learning with SMOTE for Anomaly Detection

Xiangyu Ma, Wei Shi
School of Information Technology, Carleton University, Ottawa, ON K1S5B6 Canada,
E-mail: johnnyma@cmail.carleton.ca, wei.shi@carleton.ca

Intrusion Detection Systems (IDSs) play a vital role in securing today's Data-Centric Networks. In a dynamic environment such as the Internet of Things (IoT), which is vulnerable to various types of attacks, fast and robust solutions are in demand to handle fast-changing threats and thus the ever-increasing difficulty of detection. In this paper, we present a novel framework for the detection of anomalies, which, in particular, supports intrusion detection. The anomaly-detection framework we propose combines reinforcement learning with class-imbalance techniques. Our goal is not only to exploit the auto-learning ability of the reinforcement-learning loop but also to address the dataset imbalance problem, which is pervasive in existing learning-based solutions. We introduce an adapted SMOTE to address the class-imbalance problem while remodelling the behaviors of the *environment agent* for better performance. Experiments are conducted on NSL-KDD datasets. Comparative evaluations and their results are presented and analyzed. Using techniques such as SMOTE, ROS, NearMiss1 and NearMiss2, performance measures obtained from our simulations have led us to recognize specific performance trends. In particular, the proposed model AESMOTE outperforms AE-RL in several cases. Experiment results show an Accuracy greater than 0.82 and a F1 greater than 0.824.

***Index Terms*—deep reinforcement learning, class-imbalance, adversarial strategy, anomaly detection, dynamic training, feature-selection.**

I. Introduction

CONCERNS about cyber security have emerged as a pressing issue not only in the scientific community but in society at large. It not only threatens the safety and security at the personal level but also at a much higher one [1]. For example, due to the arise of big-data era, significant amounts of biomedical data have been accumulated, which leads potential threats such as data loss, monetary theft, and attacks on medical devices and infrastructure [2]. Due to the development of Industrial wireless sensor networks (IWSN), security challenges have becoming a concern and are often dealt with intrusion detection system as a second line of defence against failure of normal network security protocols [3]. Critical Infrastructures Systems such as turbines, power plants, high-temperature energy systems, storage devices and with rotating mechanical parts are the crucial industrial systems that greatly suffer from damages on a day-to-day basis [4]. Internet of Things (IoT) refers to a network of interconnected devices that forms an emerging communications paradigm in which devices have the ability to sense the surround environment and have ways to exchange data via Internet [5]. While the smart environment provides convenience to our daily lives, such as smart homes, smart healthcare, smart devices, smart services, and smart cities, it is also very susceptible to security attacks [6]. Among all, detecting anomalies and network intrusions are the most crucial tasks. An Intrusion Detection System (IDS) refers to identifying malicious activity in a computer-related system and anomalies usually refers to abnormalities, deviants, or outliers in the data mining and statistics literature. In our daily lives, anomaly detection may appear in both research and application domains. Typical domains include fraud detection, medical anomaly detection, sensor networks anomaly detection, video surveillance, Internet of Things (IoT) big-data anomaly detection, industrial damage detection and cyber-intrusion detection [4].

Common types of intrusion detection methodologies include signature-based detection and anomaly-based detection [7]. Signature-based detection or misused detection systems focus on the idea of known threats, while anomaly-based detection systems focus on the process of comparing with normally observed events to identify significant deviations. Similar to the signature database, a profile is used to store all the normal behaviors of users, hosts, network connections, and applications. Observing the current activities, any significant deviations from the profile may be flagged as an anomaly. The main advantage of anomaly-based detection is that it allows the system to detect previously unknown attacks. Its sensitivity to any new coming threats prevents malwares which potentially consume processing resources, send large number of emails, initiate large numbers of network connections, and perform other behavior that would significantly different from the established profiles for the computer [7]. However, this sensitivity advantage can also lead high false-alarm rate which causes unnecessary panics and over reactions.

A. Problem Statement and Our Contributions

Common approaches of anomaly detection frameworks are based on supervised and unsupervised learning methods. With the existing useful information of a labelled dataset, classification becomes the most used task in supervised learning. Through an efficient learning, such as neural network (ANN), support vector machine (SVM), K-nearest neighbor (KNN), Naive Bayes, logistic regression (LR), Decision tree... etc., decent performance can occur and lead to high prediction accuracy. Nevertheless, the shortcoming of supervised learning

Manuscript received April 1, 2020.

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

is also obvious as real life datasets often come in unlabelled and labelling data manually is an expensive and time consuming task. On the other hand, unsupervised learning classifies data through unlabelled datasets, which are easy to obtain, yet the performance is usually inferior comparing to supervised learning classifiers. Hence, it is very common to see a trade-off between performance and accessibility for decision makers to take action. Most current supervised and unsupervised learning IDS are also limited to static data. As a fixed training process, they lack the adaptive learning skills for any changes to the dataset such as introduction of new labels and significant pattern changes to features. Another problem that arises for many types of datasets is due to class-imbalance. Data are said to suffer the class-imbalance problem when the class distributions are highly imbalanced [8]. In an imbalanced dataset, we might easily get an overall high accuracy, but minority classes suffer from a very low recall score. Typical examples include e-mail spam detection, medical diagnosis detection, and fraud detection. Imbalances may occur between classes, such as e-mail spam versus e-mail ham, diagnosis being positive versus negative, and fraud being normal versus abnormal. Imbalances may also occur within classes, such as e-mail spam containing a 9 to 1 ratio of advertisement versus scams, diagnosis of cancer containing 9 to 1 ratio of brain cancer versus liver cancer, and fraud detected containing a 9 to 1 ratio of trojan horse versus worms [9]. Additionally, false-positive error, in some cases, are not acceptable at all. For instance, if the cancer is regarded as positive and non-cancer as negative, failing to detect the diagnosis of cancer may cost a person' s life. Hence, it is important to have a high accuracy for the minority class as well.

To neutralize the above-mentioned problem of performance versus accessibility trade-off, Reinforcement Learning (RL) can be an efficient way to balance out their extremes. The fundamentals of RL establishes on: 1. Large and real-time datasets; 2. Agent is able to self-learn on its own: without any supervising activities during the learning process, which could self-label during the training phase based on its previous knowledge. Ideally, when data is large enough and in a real-time manner and without the consistent adjustment of labels, RL can be the better choice comparing to supervised and unsupervised learning. Furthermore, to solve the above-mentioned problem of imbalanced dataset, two conventional approaches are usually taken: the data-level approaches and algorithm-level approaches [10]. At the data-level, the original dataset is modified so that it becomes balanced enough to proceed further learning process. Re-sampling techniques are performed for this purpose and they are divided into over-sampling methods and under-sampling methods. At the algorithm-level, the objective is to make existing classifier adapt or strengthen on the learning of minority classes. In AE-RL [11], RL is integrated to generate new samples from a simulated environment and perform RL training on the simulated dataset. In addition to the conventional classifier agent, its features rely on the introduction of an environment agent, which is used to cope with the imbalance property. In this research, we set up an adversarial mechanism which provide a conflicting reward system of the two agents, so that the training dataset forms a fairly even distribution when being delivered to the classifier agent for training. This concept matches with the algorithm-level approach as mentioned above. In this paper, a data-level approach is implemented to further reduce the affect of class-imbalance.

Researchers in various disciplines have recognized that assuring sample independence is the basic assumption for any analysis conclusions. We approach this problem by applying several over-sampling and under-sampling techniques to generate artificial data, assuring no existence of duplicated data. In this paper, we present a RL framework that is built and tested on its ability to detect anomalies in networks. We use NSL-KDD dataset, on which dynamic trainings are carried out and various sampling techniques are implemented in the process to improve performance on minority classes. Our contributions are summarized as follows:

1) Establish a RL framework on anomaly detection in networks using NSL-KDD dataset.
2) Through simulation trials, we obtain comparative analysis of different over-sampling and under-sampling techniques at data-level and receive the best response based on performance.
3) Training occurs based on a recursive over-sampled/under-sampled dataset joining a feature-selection mechanism, providing a dynamic environment for performance evaluation.
4) Formulate trending for SMOTE performance on the AE-RL framework and provide comparative evaluation against AE-RL and other sampling techniques, resulting in a better performance.

II. Related Work

A. Machine Learning Approaches on Anomaly Detection

Supervised learning based anomaly detection techniques are superior in performance than unsupervised due to the labelled datasets. Semi-supervised Deep Anomaly Detection (DAD) techniques assume that all training instances have only one class label. The assumption is that points which are close to each other both in input space and learned feature space are more likely to share the same label [4]. A few works of deep learning based semi-supervised techniques for anomaly detection are presented in [12] [13]. Hybrid models are more scalable and computationally efficient due to the reduced input dimension. The downside of a hybrid model is that the learning within the hidden layer of feature extractor cannot be controlled since generic loss functions are employed instead of the customized objective for anomaly detection [4]. For unsupervised DAD, a few important assumptions are needed. First, the "normal" can be distinguished from "anomalous" regions in the latent feature space. Second, "normal" takes the majority of dataset. Lastly, outliers are identified based on intrinsic properties such as distances or densities. For example, in [14], the hidden layers of deep neural network aim to capture these intrinsic properties within the dataset. Another type of DAD is the Clustering based anomaly detection. The idea behind is to group data with similar patterns based on features extracted to detect new anomalies. [15] proposes a

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

model to obtain the semantical presentations of normal and anomaly data to form clusters.

Anomaly detection with a dynamic environment often requires an adaptive learning strategy and RL possess such character. A Monte-Carlo learning is displayed in [16] for the use of generating synthesized intrusion data based on a derived Poisson-Gamma joint probabilistic model. The proposed framework addresses the challenges of data scarcity and class-imbalance. Deep Reinforcement Learning (DRL) is a novel concept to the field, which had attracted significant interest due to its ability to learn complex behaviors in high-dimensional data space. [17] proposes a design of time series anomaly detector using DRL. The model makes no assumption about the underlying mechanism of anomaly patterns, takes away threshold settings for simplicity and adapts well to dynamic environment. [18] proposes a data poisoning attack method which takes a reverse perspective comparing to conventional IDS, showing that malicious workers, with limiting local information, still capable to find effective data poisoning attack strategies to interfere with crowd sensing systems. The proposed method is based on the DRL framework. Some typical researches have been done on the IoT data, which mostly are generated by weather stations, Radio-Frequency IDentification (RFID) tags, and IT infrastructure components [4]. [19] proposes a web attack detection system through the analysis of URLs andis deployed on edge devices. Various deep learning models are implemented and compared for evaluation. [20] proposes Vcash, a new reputation framework for tracking denial of traffic service in the Internet of connected vehicles. It borrows the idea of market trading and set up rules for the connected vehicles to follow. While restricting the malicious vehicle's spread of false message, it also performs traffic event monitoring and verification. Edge computing is mentioned in [21], introducing CloudSEC, a real-time lateral movement detection method, based on an evidence reasoning network for the edge-cloud environment. Descent performance shows that CloudSEC guarantees rapid and effective real-time attack detection. A data-driven model is proposed in [22] to achieve high-precision BGP based on deep learning methods. The model analyzes the routing behaviors without any prior knowledge.

B. Related Work on Class-Imbalance

Inappropriate evaluation metrics for generated model using imbalanced data can mislead to wrong conclusions. Common evaluation metrics include: Precision, Recall, F1 score, and Prediction accuracy. In this paper, since we are comparing our work to AE-RL, we choose to use F1 score as the standard measure. To solve an imbalanced dataset issue, countless sampling strategies had been developed throughout the years. The strategies aim to develop over-sampling and under-sampling techniques which smooth out the imbalanced property of the original dataset before any training is performed.

Random Over Sampling (ROS) [23] is the simplest techniques that works for all types of datasets. ROS extends the minority groups by duplicating them until the proportion of such minority groups reaches a predefined value. The advantage of this technique is that it does not demand any prerequisite requirements on the original dataset. However, the technique is also simple enough to affect fraud detection performances and over-sampling with replication does not always improve minority class predictions. With replication of minority groups, over-fitting may occur as the decision region becomes too specific. Hence, any new coming data that meant to belong to a minority group will be classified to others due to some trivial feature differences. On the other hand, Random Under Sampling (RUS) [23] randomly discards some majority group samples to balance the original dataset. The shortcoming appears for this technique when potential useful majority samples are dropped out during the process. Another under-sampling technique, NearMiss, is introduced early in [24] to improve prediction accuracy of minority classes. [25] later proposes a Boosted NearMiss Under-sampling algorithm on the training of SVMs (BNU-SVM), which aims to balance the dataset by adaptively updating weights over negative examples.

A different strategy to overcome the imbalance property is to generate artificial data samples, so called synthetic data generation. The Synthetic Minority Over-Sampling Technique (SMOTE) [26] proposes a technique that arbitrarily interpolates new minority samples in between several samples of that minority group, which can be found through K-nearest neighbors (KNN). This strategy avoids the over-fitting problem from ROS and still keeps the samples within the decision boundaries. A drawback for SMOTE is that the neighboring samples can be drawn from other minority groups, which may form overlapping regions among minority groups which increases additional noise. SMOTE is also not very practical in high dimensional datasets, as it does not attenuate the bias towards the classification in the majority class for most classifiers when data are high-dimensional, and it is less effective than RUS [27]. Only in cases when feature selection is applied, SMOTE can be beneficial for KNN classifiers in high dimensional datasets.

Above-mentioned works ignore to cope with both learning framework and class-imbalance concerns together, in which it is being demonstrated in our proposed model.

III. Preliminary

A. Dataset

In this paper, NSL-KDD is chosen for various reasons. First of all, it is well-organized and cleansed. Second, we have observed that this dataset still contains imbalanced data as its previous version KDD-99. NSL-KDD [28] is the refined version of KDD-99, which eliminates redundant records of the original dataset for the purpose of reducing bias towards frequent records. Moreover, the dataset has a reasonable number of entries for researchers to affordably train the complete dataset rather taking a random small portion, which makes evaluation results consistent and thus easily comparable to other research works. Hence, although NSL-KDD may not be a perfect representation of the current real network, it is still a general admitted benchmark in the IDS research field. NSL-KDD consists of 41 features, including 38 continuous

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

and 3 categorical variables, all being transformed, scaling down to the range of $[0-1]$ for the continuous ones and one-hot encoded to dummy variables for the categorical ones. After data transformation, the dataset consists of 122 features, including 38 continuous and 84 binary variables. The dataset is grouped into five major categories: NORMAL, PROBE, R2L, U2R and DoS [28]. The following are brief descriptions on each of the four attack categories:

- Denial of Service, or DoS, refers to the act of fulfilling the memory spaces and overloading the computing resources of working machines by sending unimportant information for the purpose of affecting the usage of legitimate users.
- Probing attacks, or PROBE, refers to the act of gathering information of networks for the purpose of circumventing security controls. It may be not be directly defined as an attack, but rather it is a sign of future attacks.
- Remote to Local, or R2L, refers to the act of accessing machines via sending set of packets over the network without any permission for the purpose of exploiting vulnerability to gain local access.
- User to Root, or U2R, refers to the act of obtaining full access of machines by first accessing the network resources as a normal user.

TABLE I: Attack Categories

DoS	back, land, neptune, pod, smurf, teardrop, mailbomb, apache2, processtable, udpstorm
PROBE	ipsweep, nmap, portsweep, satan, mscan, saint
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster, sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, worm
U2R	buffer_overflow, loadmodule, perl, rootkit, httptunnel, ps, sqlattack, xterm

The initial dataset comes with three types of variables: binary, nominal, and continuous. The details of feature types and label categories are shown in Table I and Table II.

TABLE II: NSL-KDD Feature Variables

Binary	Land, logged_in, root_shell, su_attempted, is_host_login, is_guest_login
Nominal	Protocol_type, Service, Flag
Continuous	Duration, src_bytes, dst_bytes, wrong_fragment, urgent, hot, num_failed_logins, num_compromised, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate

B. Reinforcement Learning

Reinforcement learning (RL) [29] is a type of machine learning that possess the ability to self-learn and develop behaviors through trial-and-error simulations with a dynamic environment. The major key components of reinforcement learning consist of the agent, the action, the environment, and the reward state. The process starts at time t with an agent taking an action, a_t, in an environment, which is being rewarded, r_t, and represented by a state, s_t. The state will be fed back to the agent for the recursive learning process.

The goal of reinforcement learning is to find a path that maximize rewards. The general formula is set up to be:

$$R_t = \sum_{k=0}^{\infty} \gamma_k r_{t+k} \text{ [30]}$$

where R_t is the total accumulated, discounted, future return that the agent receives from time t and on, $\gamma \in (0, 1]$ the discount factor, and r_{t+k} the rewards of each future time step. This long-term calculation continues to go on for future time states repeatedly and discretely. Reinforcement learning introduces the value-function by taking its expectation at state s, $\mathrm{E}(R_t \mid s_t = s)$, denoted as $V^{\pi}(s)$, is showing how good is state s, for the agent to be in. The value function depends on the policy π by which the agent chooses actions to behave. Among all possible functions, there exists an optimal value-function which has the highest value, denoted as $V^*(s) = \max_{\pi} V^{\pi}(s)$ and $\pi^* = \arg\max_{\pi} V^{\pi}(s)$ is the optimal policy which maximizes the action value achievable for state s. For convenient purpose, reinforcement learning set up a function called Q function, which takes inputs of state and action pair and outputs the value of rewards. Hence, we can rewrite the equation as: $\pi^* = \arg\max_a Q^*(s, a)$, where Q^* stands for the most optimal value for Q. According to Bellman equation, a recursive definition for optimal Q function is then defines to be:

$$Q^*(s,a) = R(s,a) + \gamma E_{s'}\left[V^*(s^{'})\right] = R(s,a) + \gamma \sum_{s' \in S} P(s^{'} \mid s, a) V^*(s^{'}) \text{ [30]}$$

where $R(s, a)$ is the immediate expected reward after performing action a at state s and $\gamma E_{s'}[V^*(s^{'})]$ is the expected, discounted, accumulated, future reward after the transition to the next state $s^{'}$.

During the process of learning, there is a trade-off of between exploration and exploitation. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The rate of exploration and exploitation are presented by ϵ and $1-\epsilon$ in reinforcement learning, respectively, ranging from 0 to 1. The exploration rate should start off in a high probability, often 1, and gradually decreases as the training goes on. When the training model is mature enough, that is reaching a decent prediction performance, the agent mostly makes decisions with its existing knowledge, based on the exploitation rate.

C. Q-learning

Q-learning was introduced by Watkins in 1989 [31]. It is a form of model-free reinforcement learning and can also be viewed as a method of asynchronous dynamic programming (DP). A function Q is introduced to represent the maximum discounted future reward when an action is performed in a state and continually optimally from that point on. The core of the algorithm is a value iteration update process, giving a learning rate for agent to learn the new coming Q value.

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a_{t+1})) \text{ [30]}$$

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

The Q value is progressively being updated, until s_{t+1} reaches the terminal state, in which the end of the episode. acts as a tuning factor, deciding the pace of learning speed, is always within the range of 0 and 1. A large α can lead to a faster learning process, yet may loses some information on the way. Similarly, verse versa.

D. Deep Q-learning

Although Q-learning is a powerful algorithm for policy selection, it does not have the ability to estimate values of unseen states. Hence, for an infinite state space, where environment setup may be changed, some Q-values cannot be calculated. Deep Q Network (DQN) is introduced to deal with this problem of lack of generality. A neural network is implemented, where the currents are the input and the estimates of Q values for each actions are the output. [17] The target Q-value for Q-learning is:

$$r_j + \gamma_{a'} Q(\phi_{j+1}, a^{'}; \theta^{-}) \text{ [32]}$$

where ϕ is equivalent to state s and θ is the parameters in the Neural Network. Since we are approximating Q-values, the goal is to minimize the error between target Q-value and the Q-value output from the network:

$$Loss = (y_j - Q(\phi_j, a_j; \theta))^2 \text{ [32]}$$

A gradient descent is performed to find the minimal error.

IV. AESMOTE Framework Description

A. General Description

As mentioned previously, RL has two advantages over conventional supervised learning: easy to deal with large and real-time datasets and has the ability to learn on its own without any supervision. This section will introduce the general process of RL and the detailed structure of AE-RL along with techniques that associate with the class-imbalance concern.

Instead of giving rules, correcting input/output pairs and finding similarities and differences between data points, reinforcement learning emphasizes on the performance, which bases on rewards and punishments as instructions for positive and negative behavior. The reward system is assigned to a software agent, who constantly interacts with the environment and learns from the acquired rewards and punishments. Supervised and unsupervised learning are usually one-shot, myopic, considering instant reward; while reinforcement learning is sequential, far-sighted, considering long-term accumulative reward [33].

In a RL framework, the agent, the state, the action, the environment, and the reward are components of the learning loop.

TABLE III: RL Components (Classifier Agent)

Components	Corresponding Implementation
Agents	Classifier
Environment	Training dataset
State	Current line of data
Action	Making prediction of current label type
Reward	Correctness of prediction

Table III defines the corresponding matches of RL components to our real dataset. During the active RL process, for every incoming data, it is noted as the current state. Making a prediction by the classifier agent would be regarded as taking an action. Receiving a feedback of whether the prediction is correct is defined as the reward. Correct predictions receive a value of +1 and incorrect predictions receive a value of -1.

In this paper, Q-learning is chosen for the base learning algorithm. As mentioned previously, Q-learning finds the maximum discounted accumulated reward for each state-action pair. In other word, for each incoming line of data, Q function will find all rewards for every possible action and accumulate the best ones in record. The Q-learning functions as follows:

$$Q^{new}(s_t, a_t) \leftarrow$$
$$(1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a_{t+1})) \text{ [30]}$$

In this equation, s_t represents the current state, a_t represents the current action, $s_t + 1$ represents the next state, $a_t + 1$ represents the next action, γ represents the discount factor, α represents the learning rate, r_t represents the current reward, and Q^{new} represents the updated Q value. In this paper, values of 1.0 and 0.001 for α and γ are assumed, respectively, based on the assumption that all individual data are independent.

Now, the question comes to which data we should give for the agent to train. If, let's say, the original training set is 100 times larger, running through the complete dataset would be non-realistic. Even if we have enough computation power to train the complete dataset, the imbalance problem is still existence. Hence, a mechanism is constructed to handle this problem, noted as the environment agent. Similar to the classifier agent, the environment agent also performs Q-learning for every incoming data. The difference is that it accumulates opposite rewards as the classifier agent. For instance, when the incoming NORMAL data is predicted to be *normal* by the classifier agent, the environment agent receives a negative reward. The essential idea behind the environment agent is to learn the prediction performance of classifier agent and select the most appropriate categories of data for classifier agent to train next. Hence, the classifier agent is always forced to train the most difficult samples at the moment, which balances out the distribution of training samples. With the cooperation of environment agent, a more balanced dataset is simulated, accuracies may improve, especially for the minority classes such as U2R and R2L.

TABLE IV: RL Components (Environment Agent)

Components	Corresponding Implementation
Agents	Environment
Environment	Training dataset
State	Current type of data
Action	Making prediction of incoming label types
Reward	Correctness of the classifier's prediction (opposite reward)

As described in Table IV, the process begins with the current state being the current type of data, the current action being making prediction of the type of next line of data for training based on its policy, and receiving opposite amount of reward as classifier agent in memory.

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

One important thing to note is that there are two different epsilon value for the two agents as these values decide the learning exposure of agents to the incoming data.

Algorithm 1: Epsilon - Greedy

Initialize $Q(s, a)$, the Q-function;
Initialize $K > 1$, the number of available actions;
Initialize $c > 0$ and $0 < d < 1$, coefficients;
Initialize $\epsilon \in (0, 1]$;
for $t = 0$ *to* N **do**
  $\epsilon_t = \min\{1, \frac{cK}{d^2 t}\}$;

$$a_t = \begin{cases} argmax_\pi Q(s,a) & \text{with probability } 1 - \epsilon_t \\ \text{a random action} & \text{with probability } \epsilon_t \end{cases}$$

return a_t

As we can see from Algorithm 1, ϵ first start off being equal to 1 and is dynamically changing in a descending trend as t increases. Actions are then calculated by taking the piecewise choice based probability ϵ. Two agents both have their epsilon estimates and are different for each case. The lower bound of epsilon for the classifier agent is relatively low as the model is becoming more and more stable, which is set to 0.01. We set this low value because classifier agent is our main agent as it's "the final decision maker" and we want it to have a high exploration rate. On the other hand, the lower bound of the epsilon for the environment agent is set as a hyperparameter [11], with an ideal value of 0.8. The reason for this value to be large is that there is no need for the environment agent to be absolutely accurate since it's based on an unstable factor. Additionally, all these values are selected from trial-and-error process.

B. Over-sampling and Under-sampling methods

Now, the question comes to the existence of duplicate data when selecting samples from the original dataset D. This problem may appear trivial for the majority classes such as NORMAL, DoS, and PROBE, but when dataset is small enough, like R2L and U2R, which only has around 1000 and 50 lines of data, respectively, it is very likely that data is being selected more than once. Hence, over/under-sampling techniques are applied and compared.

To begin, we need to initialize the value K as the K-Nearest Neighbor, which will be used in all the following techniques. For Over-sampling, first we set R2L and U2R as our minority classes. According to [34], some key features such as "number of file creations" and "number of shell prompts invoked" can be carefully looked when over-sampling U2R data and "duration of connection", "service requested", and "number of failed login attempts" can play an significant role for R2L data. The essential idea is to keep the same values for these features when generating new dataset, while other unimportant features are randomly generated. The next step is applying different over-sampling techniques. In this paper, over/under-sampling techniques SMOTE, ROS, NearMiss1, and NearMiss2 are applied and compared. These techniques are chosen because

Algorithm 2: over/under-Sampling Algorithm

Import dataset D;
Initialize dataset D_M;
Initialize K for K-Nearest Neighbor;
Initialize key_feature_list = [];
Over-sampling:
for *each incoming data* **do**
  if *minority* **then**
    while *feature not in key_feature_list* **do**
      Apply over-sampling techniques include:
       ROS and SMOTE;
      D_M = D_M.append(new data);
    end
    D_M = D_M.append(mean(data));
  end
end
Under-sampling:
for *each incoming data* **do**
  if *majority* **then**
    Apply under-sampling techniques include:
     NearMiss-1 and NearMiss-2;
    D_M = D.remove(data);
  end
end

of their simplicities of implementation and the fact that they take completely different approaches, which results in easier comparisons.

1) Random Over Sampling (ROS)

Random Over Sampling (ROS) is the most naive approach to generate new samples. The new samples are randomly duplicated in the minority classes with replacement. They are referred "naive resampling" due to the fact that they assume nothing about the nature of data and no heuristics are used. The advantage, however, is that it is simple to implement and fast to execute.

2) Synthetic Minority Over-Sampling Technique (SMOTE)

Synthetic Minority Over-Sampling Technique (SMOTE) proposes the idea of generating synthetic samples based on existing ones. Based on the "feature space" of data, the minority class is over-sampled by generating new samples on the line segments formed by the endings of its K nearest neighbours [26]. K is chosen depending on the amount of over-sampling required. On each line segment, the newly generated samples can be determined by multiplying the difference between the feature vector and its nearest neighbour by a random number ranged from 0 to 1. This approach allows generated synthetic samples to follow certain feature patterns and forces the boundaries of minority classes to become more general.

3) NearMiss Undersampling

On the other hand, under-sampling techniques focuses on reducing the size of majority samples. In NSL-KDD dataset, we set NORMAL and DoS as the majority classes since they take up to 53.46% and 36.46% of the whole dataset, respectively. The NearMiss [25] [24] under-sampling technique aims on the distance measures between these majority classes and a few

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

specific minority samples. Two types of NearMiss approach are presented here:

NearMiss 1: Keep only the majority samples for which has smallest average distance to the N nearest minority samples.

NearMiss 2: Keep only the majority samples for which has smallest average distance to the N farthest minority samples.

After pre-processing the training data, we can put it into the RL structure.

C. Main Algorithm

In addition to AE-RL, our algorithm provides a synthetic, more balanced, and more packed dataset before any learning process. Moreover, some features are added into the main algorithm for a more data-specific analysis. Every step learning is based on episodes and the number of episodes is based on the computation power of the machine. A feature-selection is applied during the process, in which keeping the important characteristics of the newly generated samples in order to reduce biases. The training dataset, D, is modified by performing over/under-sampling techniques and the modified dataset, D_M is constantly being reset at the start of every episode so that brand new simulated datasets are generated and used for training the agent's adaptive skills.

Algorithm 3: Main Algorithm

```
Import dataset D;
Initialize Q_c(s_i, a_ci) arbitrarily;
Initialize Q_e(s_i, a_ei) arbitrarily;
for each episode do
    Reset D_M;
    s_0 = A random sample of dataset;
    for each sample within episode, i ϵ [0, N] do
        Memory_temp = [];
        D_M = Apply over/under-sampling techniques
          to s_i of dataset D;
        Classifier agent: choose action a_ci based on
          Q_c(s_i, a_ci);
        Environment agent: choose action a_ei based on
          Q_e(s_i, a_ei);
        Obtain rewards: r_ci, r_ei;
        Derive next state: s_{i+1} = S(a_{e(i+1)}), where
          S(a_{e(i+1)}) is randomly selected from all
          samples whose label is a_{e(i+1)} of the
          simulated dataset D_M;
        Update state, action, next_state, reward;
        while s_{i+1} is in Memory_temp do
            Reselect s_{i+1} from all samples whose label
              is from a_{e(i+1)} of D_M;
        end
        Update Memory_temp by appending s_{i+1};
    end
end
```

Referring to Algorithm 2, the main algorithm starts by importing the original dataset. The Q values for the current data with the prediction of classes based on an accumulated policy, defined as $Q_c(s_i, a_{ci})$ and $Q_e(s_i, a_{ei})$, respectively, are arbitrarily initialized as they will eventually be auto adjusted by the RL process. For every episode, a random data is selected as the initial state before the RL process. Then, for every incoming data, the dataset is consistently being updated by performing over/under-sampling techniques to s_i, the current data's sub-class (eg. normal, back, ipsweep, ftp_write, and buffer_overflow) in keeping the ratio to its major class (eg. NORMAL, DoS, PROBE, R2L, and U2R). Next, RL components are discovered for both classifier and environment agents. Following their given rewards, classifier will make prediction of the current class type, a_{ci}, based on its policy, $Q_c(s, a)$, while environment provides the next training data, a_{ei}, based on previous classifier's performance, $Q_e(s, a)$. In the case of NSL-KDD, a_{ci} is an element of NORMAL, DoS, PROBE, R2L, and U2R and a_{ei} is an element of the features in the training dataset. Similarly, r_{ci} is the positive or negative reward corresponding to classifier's correct or incorrect classification, respectively. Meanwhile, r_{ei} takes opposite rewards as the classifier agent. The next state, s_{i+1}, is derived by randomly picking a sample from $a_{e(i+1)}$, the resulting class decided by the environment agent. Meanwhile, a memory checker is constructed in parallel to eliminate the existence of duplicated data during the data selection phase.

The algorithm progresses along with a decreasing epsilon-greedy policy. Both agents start off with a high exploration rate, ϵ = 1, indicating the fact that they had no previous knowledge of the dataset. Then, their rates gradually decreases as the learning goes on and converges to lower bounds of ϵ_c = 0.01 and ϵ_e = 0.8, for the classifier agent and environment agent, respectively. ϵ_c is set relatively low since an ideal predictability for the classifier agent is to have the entire control of the training set based on existing knowledge. On the other hand, the environment agent requires a more arbitrary distribution method of data in cases of uncontrollable learning patterns from the classifier agent.

D. Models and Error Reduction

Let's recall that Bellman equation relating to Q functions of consecutive steps is:

$$Q^*(s,a) = R(s,a) + \gamma \max_{a'} Q(s', a') \text{ [30]}$$

Through a temporal difference learning which iteratively approximates the Q values at each time step, a loss function can be derived so that we can minimize the mean squared error for more accurate estimations:

$$L = \tfrac{1}{2}[R(s,a) + \gamma \max_{a'} Q(s', a') - Q(s', a')] \text{ [30]}$$

Since we can never be accurate in estimating the actual $Q^*(s, a)$, in practice, we apply the gradient descent for every iteration to find the local minimum of loss and eventually arrives at the global minimum of the loss function L. In addition, we use Huber loss as the base for the actual loss function to apply robust regression, which consists of a piecewise function that is quadratic when the parameter is small and becomes linear beyond a threshold. The advantage is that it eliminates sensitivity to outliers (i.e., explosive behaviours).

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

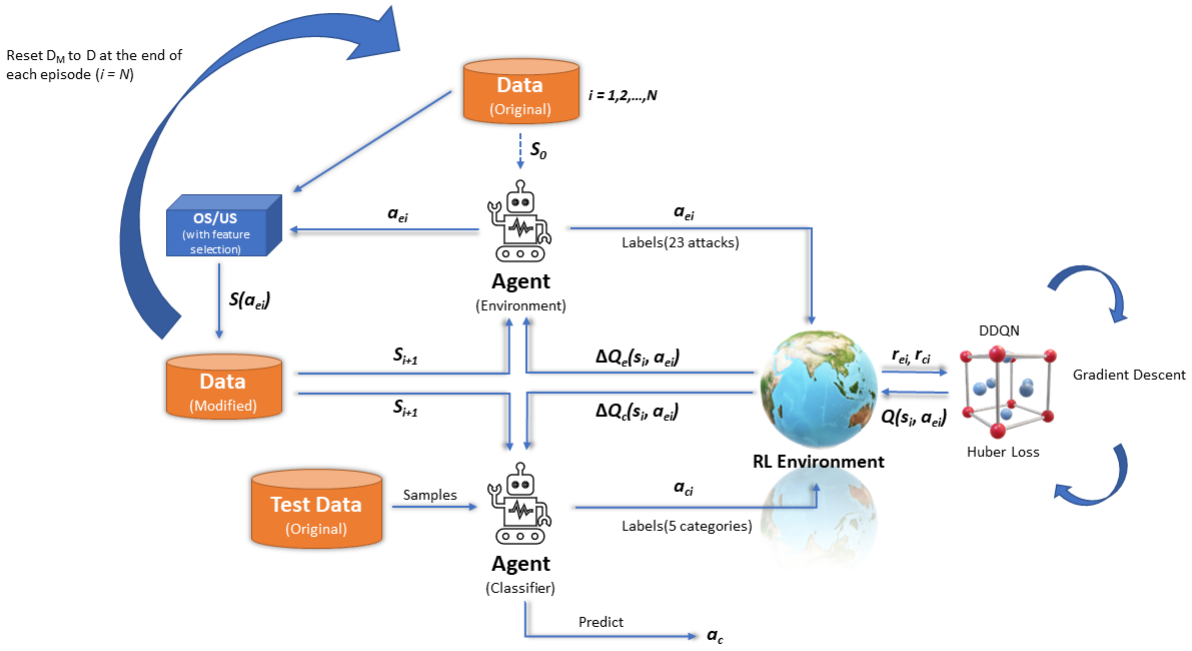

Fig. 1: Algorithm Structure

1) Double Deep Q-Network

Consider the Q-function, more specifically:

$$\max_{a'} Q(s^{'}, a^{'})$$

Taking the maximum Q-value among all possible actions at a given state is our current standard for getting the optimal value of the learning process. However, this would cause an overestimation of the actual Q-values. For instance, suppose the true label for a specific data is R2L, if the decision-changing features are distributed some above and some below the decision threshold, by taking the maximum of these estimates would always lead to the choice of one above the threshold.

To deal with this concern, we choose the Double Deep Q-Network (DDQN) for our model structure. The DDQN structure involves using two separate Q-value estimators for action-value estimation. They function as constraints to each other by selecting actions using opposite estimators. Thus, the maximization bias can be disentangled. The details of DDQN algorithm are shown in Algorithm 4.

After $Q_c(s, a_c)$ is randomly initialized, for each incoming data, the Q-values are being updated. As opposed to a regular DQN, where actions are chosen by the innate Q-learning policy, DDQN always chooses the optimal action based on the $Q_{\theta'}$ function, which is a function approximation. Then, we minimize the mean squared error between $Q_c(s, a_c)$ and $Q_{\theta'}$ for a better action evaluation. Lastly, $Q_{\theta'}$ is slowly being transformed to $Q_c(s, a_c)$ through Polyak averaging, where $\theta^{'}$ refers to the target network parameter, θ refers to the primary network parameter and τ refers to the rate of averaging which is set to 0.01 as a default hyper-parameter [35].

Algorithm 4: DDQN

Initialize $Q_c(s_i, a_{ci})$ arbitrarily;
for *each sample within episode,* $i \in [0, N]$ **do**
  Store $(s_i,\ a_i,\ r_i,\ s_{i+1})$ in replay buffer;
  Compute target Q:
    $Q_c(s_i, a_{ci}) \approx$
    $r_i + \gamma Q_\theta(s_{i+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a^{'}))$;
  Perform Gradient Descent:
    $(Q_c(s_i, a_i) - Q_\theta(s_i, a_i))^2$;
  Update target network parameters:
    $\theta^{'} = \tau\theta + (1 - \tau)\theta^{'}$;
end

E. A Walk-through

Figure 1 illustrates a detailed procedure of our algorithm, which concludes all parts mentioned above. In this subsection, we walk-through the algorithm with one simulation trial. Assume a majority class sample, say normal (NORMAL), is selected as the initial state from the original training set D and is being passed to the environment agent (EA). Without any previous knowledge about the data, EA makes a prediction, a_{ei} (23 labels) based on an arbitrary Q function, $Q_e(s_i, a_{ei})$, say neptune (DoS) and on one hand performs over-sampling/under-sampling techniques (OS/US) to produce a modified dataset, D_M and the other passes into the RL environment for modelling. Then, a neptune sample from D_M is randomly selected and passed to both the EA and classifier agent (CA) as the next state. Meanwhile, Q functions are updated for both EA and CA by going through a series of

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

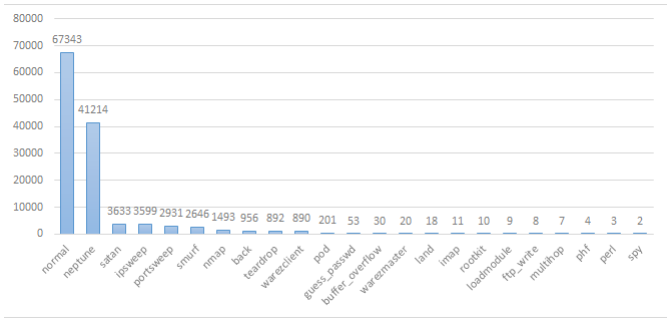

Fig. 2: Subclasses

modelling with DDQN, gradient descent, and loss reduction in the RL environment. Receiving the state and updated Q functions, EA and CA could consistently take actions for every incoming data. At the end of every episode, D_M will be reset to the original training set, where the dynamic training occurs by training a dataset that had new data samples added dynamically. After training the specified total number of episode, CA makes prediction of the final classes (5 labels) on the testing set.

V. Experimental Evaluation

A. Dataset Description

The dataset is split into a training set of 125973 samples and a testing set of 22544 samples. When comparing the training and testing set, a few facts raise concerns:

1) Both datasets have only 21 labels in common.
2) 2 labels are unique that only appear in the training set and 17 labels are unique that only appear in the testing set.

This implies that a lot of testing cases are not being trained at all in the training set, which would be a challenge as the agent has to be accurate enough to generalize the categories in order to do well against the untouched samples. However, this problem can be solved as follows.

As Figure 2 demonstrated, the subclasses normal and neptune takes the majority portion of the dataset, 53.45% and 32.71%, respectively, whereas the remaining subclasses only take 13.84%. Thus, the dataset is extremely unbalanced. Now, to make the dataset meaningful, these 23 labels are categorized into five categories: NORMAL, DoS, PROBE, R2L, and U2R [28]. (Further details can be found in Table I) The advantage of this action is that even for the new samples in the test set, the agent is able to predict its category, rather its subclasses.

As Figure 3 demonstrated, the imbalanced property becomes more evident as the biggest class NORMAL contains 53.45% while the smallest class U2R only contains 0.04% of the whole dataset. Figure 4 displays the architecture of input and output on which both agents are based. In the architecture, two neural networks are implemented with shallow layer numbers of 3 and 5 for environment and classifier agent, respectively. The input of both agents are equivalent as they take in all the features in the training data for learning. Their outputs, however, are different as they serve for different purposes. The objective for environment agent is to summarize the most

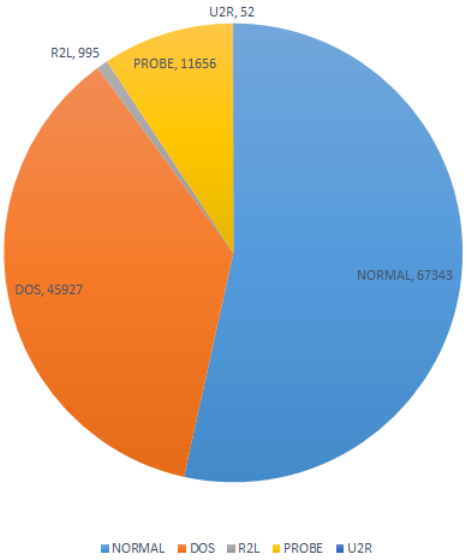

Fig. 3: Distribution of Categories

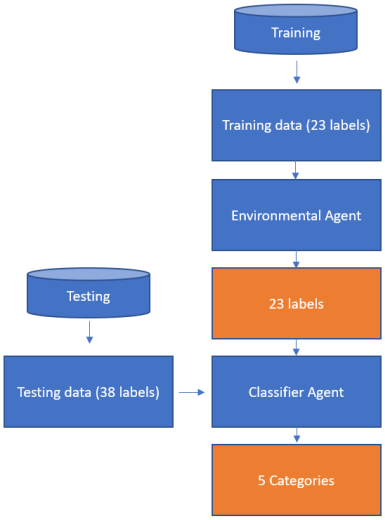

Fig. 4: Input and Output

needed samples for training at the moment, which requires the output to be as specific as it can. Hence, the output of environment agent is the 23 labels. On the other hand, classifier's objective is to produce a meaningful prediction of attacks so that intrusion defender can take immediate actions. As mentioned previously, data is categorized into 5 categories according to their natures. Hence, the output of classifier agent is the 5 categories. Another advantage of this different output strategy is that it has the ability to cope with the untrained labels. The classifier agent would always make the most optimal estimation based on its learned characteristics of closest category.

B. Evaluation Metrics

When evaluating the performance of the prediction, two evaluation metrics are used: classification accuracy and F1-

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

score.

Accuracy is the most intuitive performance measure as it is simply the ratio of correctly predicted observations to the total observations. Yet, one shortage of using accuracy as measurement is that it is only good when the false positive and false negative rates are relatively close to each other. Otherwise, it should only be considered as a reference parameter. F1-score is the combined calculation of precision and recall. By taking the weighted average of both measurements, it takes both false positives and false negatives into account. Hence, it is more reliable to use for imbalance datasets.

VI. Parameter Setup

When performing the training, several major parameters had been examined and determined to be certain values which is ideal for the model. As training begins, the exploration rates, ϵ, are set to lower bounds of 0.01 and 0.8 for the classifier agent and the environment agent, respectively. Since ϵ_c is used to progress the classification tasks, hence it is strictly following a regular bound. On the other hand, ϵ_e is used to choose tasks for training, hence as long as the distribution is relatively balanced, it is encouraged to be more explorable. During the Q-learning, the learning rates for both agents are set to 0.2 and the discount factor is set to 0.001. The layers of the neural network, l_c abd l_e are set to 5 and 3 for the classifier and environment agent, respectively. The number of nearest-neighbors that is used to generate synthetic data, k, is set to a value greater equal to 3. Since several subclasses have a small quantity of samples, such as multihop (7), ftp_write (8), loadmodule (9), imap (11), and land (18), a large number of nearest-neighbors would produce bias results. For cases such as spy (2), perl (3), and phf (4), we select k as 1. Through repeatedly testing, we set the number of training episodes and iterations per episode lower bounds of 100 since the performance is reaching its optimal after these values.

TABLE V: Training Parameters

Parameter	Description
ϵ_c	Exploration rate for classifier agent, > 0.01
ϵ_e	Exploration rate for environment agent, > 0.8
α_c	Learning rate for classifier agent, = 0.2
α_e	Learning rate for environment agent, = 0.2
γ	Discount factor, = 0.001
l_c	Hidden layers for classifier agent's neural network, = 5
l_e	Hidden layers for environment agent's neural network, = 3
k	Number of Nearest-Neighbors for SMOTE, ≥ 3
$num\_episodes$	Number of training episodes, ≥ 100
$iterations\_episode$	Number of iterations per training episode, ≥ 100

As demonstrated in Figure 5 and Figure 6, we observe that both F1 and accuracy have an increasing trend from 10 episodes to 100 episodes. The performance after 100 episodes becomes stable around 0.82 for both F1 and accuracy. Hence, the number of episode for training is tuned to be greater than 100 episodes to achieve best results.

Figure 7, 8, and 9 display the changes as number of episodes increase. At 10 episode, the class U2R is being overestimated. This is due to the fact that we forced the balanced learning

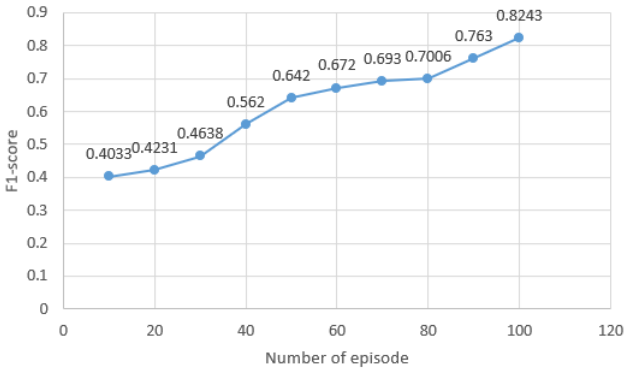

Fig. 5: F1 scores for different number of episodes

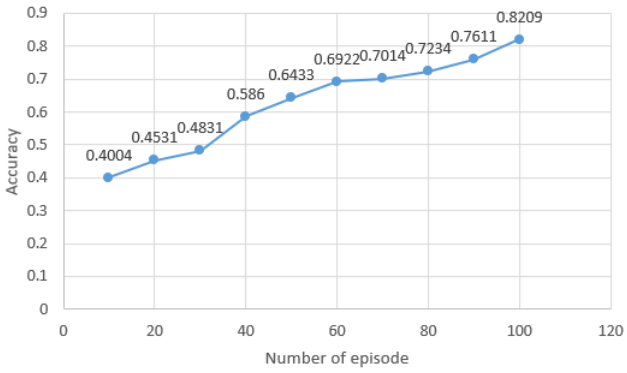

Fig. 6: Accuracy scores for different number of episodes

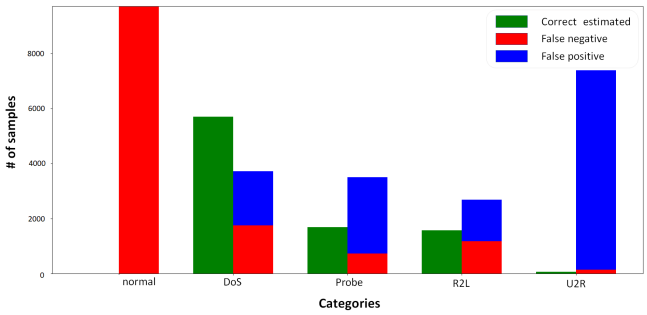

Fig. 7: F1 score: 0.4033 for 10 episodes

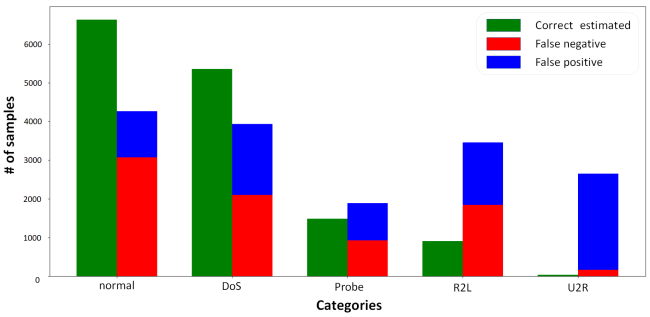

Fig. 8: F1 score: 0.642 for 50 episodes

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

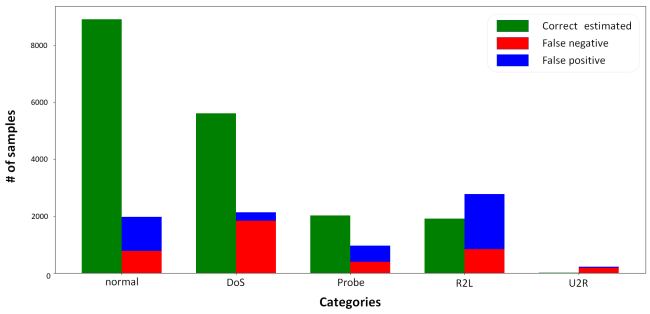

Fig. 9: F1 score: 0.824 for 100 episodes

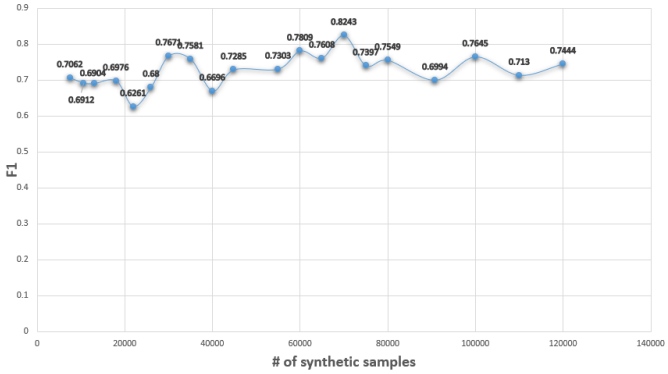

Fig. 10: F1-score Trending on SMOTE

strategy onto the classifier agent and it's not mature enough to identify the differences between classes yet. However, it is showing a sign that our mechanism starts to kick in as the agent is treating all classes equivalently. At 50 episode, the agent is already mature enough to clearly identify NORMAL class, reaching a descent true positive rate, yet still unclear on the minority parts. Finally, at 100 episode, the boundary between the classes is becoming more clear and achieves an F1 of 0.824.

VII. SMOTE Performance

In this section, we are going to demonstrate the performance of SMOTE on the AE-RL framework. As note, the purpose of SMOTE is to generate synthetic samples in accordance to its own data patterns so that minority classes have enough samples to be transferred to the classifier agent for training. Considering the highly unbalanced property of the dataset, the metric F1 will be used to measure performance [36]. As mentioned before, F1 takes both precision and recall into consideration, which assures both false-positive and false-negative into account. In our case, false-positive and false-negative rates are both crucial measures. For instance, a NORMAL class is predicted to be an intrusion attack, say DoS, can lead to false-alarm. F1-score can be further divided down to "one vs. rest" and "aggregated". Considering the "one vs. rest" approach, the NORMAL class is the "one" while all other four classes belong to the "rest", which results in a binary classification. On the other hand, the "aggregated" refers to all classes being equivalent. In this paper, we use the aggregated F1-score as our measure, which takes the weighted average of all F1-scores. For result comparison, we are interested in the F1-score trending on SMOTE performance.

Equal amount of samples are generated for both minority classes U2R and R2L. For instance, if we generate 20000 samples, 10000 samples will be generated for U2R and R2L, respectively. To test the trending, 20 trials had been recorded for presentation. As shown in Figure 10, we can observe that the algorithm AESMOTE's performance, although fluctuating, but demonstrating an increment, reaching a peak of F1 = 0.8243, when the sample size is between 5000 to 70000. After a closer inspection of the result data, we observe that the implementation of SMOTE had putting on some negative effects at first, causing the F1-score averaging around 0.7 while the original AE-RL has an average of 0.78. This is due to the instability of newly generated samples. Although, theoretically, the synthetic samples are "within" its neighbours, but they are still not as reliable since they might produce new non-realistic pattern which could affect the real data's performance. However, as more samples are being generated, this non-realistic pattern becomes stable enough and slowly separates itself from the real data's pattern. The advantage of this newly created pattern is that it could become useful to deal with the non-trained labels in the testing set. This is also the reason that an outstanding score of 0.825 could be found in the trend as opposed to the flat average of 0.78 by AE-RL.

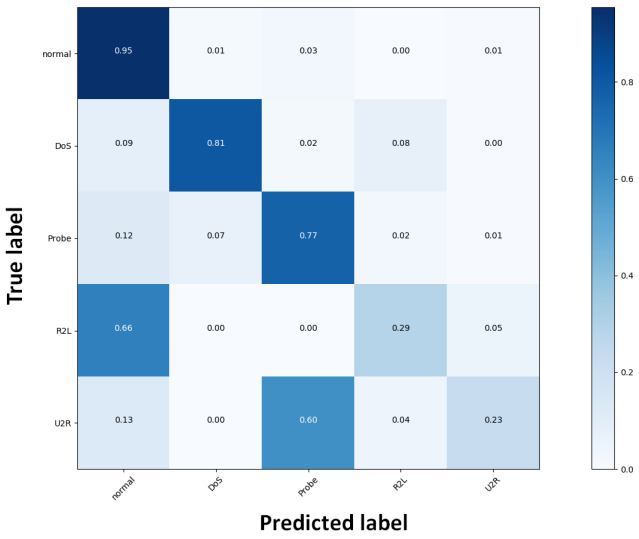

Fig. 11: Confusion Matrix for AE-RL

Looking at the confusion matrix for both algorithms in Figure 11 and Figure 12, we observe that the true-positive rate for the minority class R2L has a significant increase from 0.29 to 0.69, showing an improvement due to the over-sampling mechanism. As we look into the subclasses in Figure 13 and Figure 14, we can see that the number of correct estimates for the minority class R2L has a significant increase for AESMOTE comparing against the original AR-RL.

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

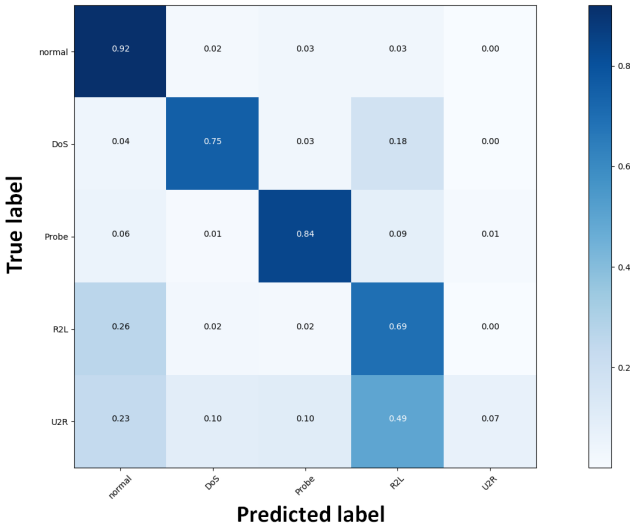

Fig. 12: Confusion Matrix for 70000 samples with SMOTE

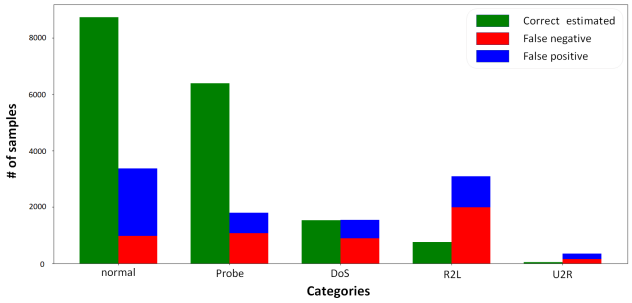

Fig. 13: AE-RL, F1 = 0.78

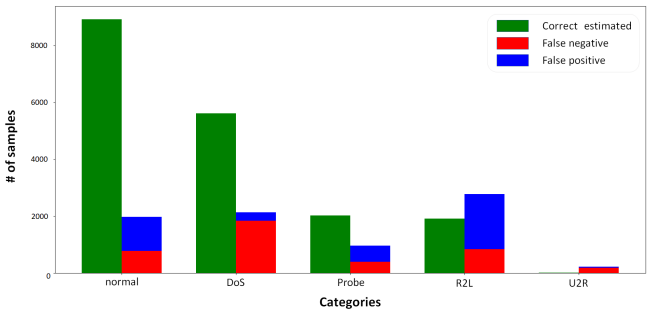

Fig. 14: AESMOTE, F1 = 0.8243

VIII. Performance on Other Methods

Other than the SMOTE sampling method, we've also tried implementing Random Over Sampling (ROS) [23], NearMiss1 and NearMiss2 [25] [24] under sampling method. Results are obtained in Table VI.

As note, the main idea of ROS is duplicating existing data, which is against our initial purpose. After several trials, the highest F1-score that we obtained were 0.7083. Theoretically, this method is considered the most unreliable method, although it does have the potential of discovering new data patterns, its performance is relatively non-stable. For under-sampling

TABLE VI: Across Algorithms

Method	F1-score	Acc.	Prec.	Rec.	Time (sec.)
AE-RL	0.78	0.7744	0.7616	0.7744	350
AESMOTE	0.8243	0.8209	0.8411	0.8209	2000
ROS	0.6932	0.6977	0.7112	0.6932	1000
NearMiss1	0.7602	0.7611	0.7795	0.7611	195
NearMiss2	0.7865	0.7805	0.8027	0.7805	84

techniques such as NearMiss1 and NearMiss2, the NORMAL and DoS are considered as the majority classes. Yet again, this does not solve the duplicated data problem since it only applies a decrement of the majority classes' data size. However, they have the advantage of a faster runtime of 195 seconds and 84 seconds, respectively and only 10 episodes are required for training due to the data size reduction. Comparing to AE-RL, AESMOTE has a longer runtime due to the addition of sampling techniques and feature-selection mechanism, yet they have brought up the F1-score to 0.8243.

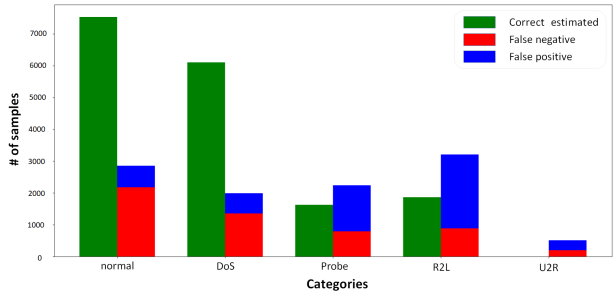

Fig. 15: NearMiss1, F1 = 0.7602

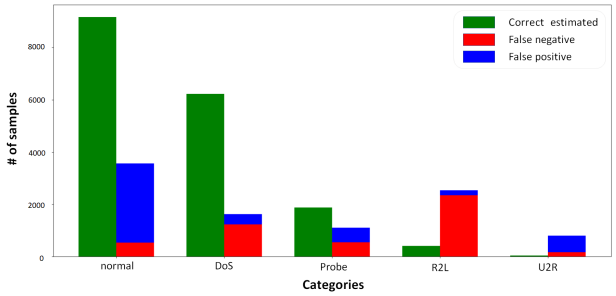

Fig. 16: NearMiss2, F1 = 0.7856

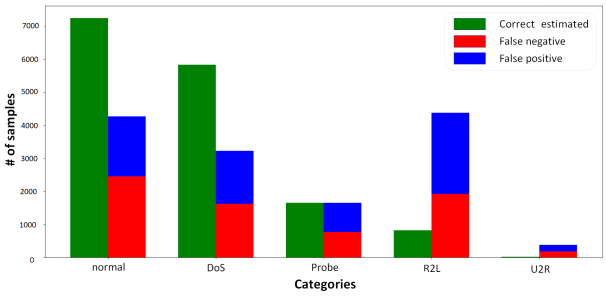

Fig. 17: ROS, F1 = 0.6932

Figure 15, 16, and 17 shows the detailed prediction of each class. We can observe that all of them have weak perfor-

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

mances on minority classes with low true positive rates. More specifically, NearMiss1 has lower *normal* prediction accuracy than NearMiss2. This is due to the fact that NearMiss1 keeps the majority samples that are closest to the nearest minority samples, which means rather keeping the whole identify of the majority class, it only keeps a cluster of samples. Simultaneously, since majority classes are in a cluster form and does not cover a wide range of area, more predictions are made for the minority classes, which results in a higher number of correct estimates. On the other hand, NearMiss2 has descent performance on the majority classes NORMAL, DoS, and PROBE, but lower number of correct estimates in R2L. This is due to the fact that NearMiss2 keeps the majority samples that are closest to the furthest minority samples. Instead of forming a cluster-like group, it forms a line-like shape. The advantage is having more accurate majority classes' predictions, but gives less predictions on the minority classes. Lastly, we can observe that ROS suffers in performance from both majority and minority classes. The fact that duplicating minority samples leads to an inferior result.

IX. Conclusion

IDS is a critical service that monitors networks for malicious activities such as security attacks. Difficulties have been imposed due to the network's complex and dynamic environment, which could severely affect the performance of existing IDS software. These difficulties include unbalanced, complex and asymmetric datasets. This paper modifies the framework of AE-RL by imposing sampling techniques and specifically investigates the performance of joining SMOTE with the AE-RL framework. RL algorithms are mostly successful in other domains such as videogames, strategy games, robotics, finance, resource management, chemistry, web system configuration, etc. Yet it is very new to IDS because rewards are generally difficult to be defined.

In this paper, we not only adopt RL to detect anomalies including anomaly-based intrusions in networks, but also improves the class-imbalance problem which is pervasive to datasets in many domains. Our algorithm takes a labelled dataset as input, then provides a RL framework based on an adversarial strategy. Sampling techniques are implemented for better data selection in order to achieve the best possible classification results. Additionally, we propose a feature selection mechanism during the data selection phase so that the simulated data still capture their main characteristics.

In summary, we present a joint framework of supervised learning, adversarial RL and sampling techniques, which results in an increasing performance of intrusion prediction. Our proposed model leverages a thorough training process on dynamic data environment. The experiment results obtained provide a thorough performance trending analysis of AESMOTE to display the optimal number of generating samples in practice and a comparative analysis of AESMOTE with AE-RL, fully displaying the benefit of our model. Finally, the comparative evaluation results demonstrate that the proposed AESMOTE has a better prediction performance than other sampling techniques such as ROS, NearMiss1 and NearMiss2.

For future work, we plan to expand the work load of environment agent. First, we will introduce "difficulty levels" to the dataset using existing supervised learning techniques. Based on its performance, the "difficulty levels" will be further distributed into multiple partitions. By setting up multiple environment agents, we provide a multi-adversarial strategy for data selection. The advantage of this idea is to further identifies the "weakness" of the current classifier agents, which further balances the dataset.

Acknowledgment

We gratefully acknowledge the financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grants No. RGPIN-2015-05390.

References

[1] G. Fernandes, J. Rodrigues, L. Carvalho, J. Al-Muhtadi, and M. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, no. 3, pp. 447–489, 2019.
[2] M. Jaladi and N. Ghaffarzadegan, "Cybersecurity in hospitals: A systematic, organizational perspective," *Journal of Medical Internet Research*, vol. 20, no. 5, p. e10059, 2018.
[3] D. Ramotsoela, A. Abu-Mahfouz, and G. Hancke, "A survey of anomaly detection in industrial wireless sensor networks with critical water system infrastructure as a case study," *Sensors*, vol. 18, no. 8, p. 2491, 2018.
[4] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: a survey." https://arxiv.org/abs/1901.03407, 2019.
[5] M. Elrawy, A. Awad, and H. Hamed, "Intrusion detection systems for iot-based smart environments: a survey," *Journal of Cloud Computing*, vol. 7, no. 21, 2018.
[6] J. King and A. Awad, "A distributed security mechanism for resource-constrained iot devices," *An International Journal of Computing and Informatics*, vol. 40, no. 1, 2016.
[7] K. Scarfone, "Guide to intrusion detection and prevention systems (idps)," *Computer Security Resource Center*, 2012.
[8] C. X. Ling and V. S. Sheng, *Class Imbalance Problem*. Springer, Boston, MA, 2011.
[9] N. Japkowicz, "Concept-learning in the presence of between-class and within-class imbalances," *Canadian AI 2001: Advances in Artificial Intelligence*, vol. 2056, pp. 67–77, 2001.
[10] J. L. Leevy, T. M. Khoshgoftaar, B. R. A., and S. N., "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 42, 2018.
[11] G. Caminero and B. Lopez-Martin, M. Carro, "Adversarial environment reinforcement learning algorithm for intrusion detection," *Computer Networks*, vol. 159, pp. 96–109, 2019.
[12] B. Kiran, D. Thomas, and R. Parakkal, "An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos," *Journal of Imaging*, vol. 4, no. 2, p. 36, 2018.
[13] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma, "Su-ids: A semi-supervised and unsupervised framework for network intrusion detection," *Cloud Computing and Security*, vol. 11065, pp. 322–334, 2018.
[14] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLOS ONE*, vol. 11, no. 4, pp. 1–31, 2016.
[15] G. Yuan, B. Li, Y. Yao, and S. Zhang, "A deep learning enabled subspace spectral ensemble clustering approach for web anomaly detection," (Anchorage, USA), pp. 3896–3903, 2017.
[16] H. Zhang, X. Yu, P. Ren, C. Luo, and G. Min, "Deep adversarial learning in intrusion detection: A data augmentation enhanced framework," *arXiv*, 2019.
[17] C. Huang, Y. Wu, Y. Zuo, K. Pei, and G. Min, "Towards experienced anomaly detector through reinforcement learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, (Hilton New Orleans Riverside, USA), 2018.
[18] M. Li, Y. Sun, H. Lu, S. Maharjan, and Z. Tian, "Deep reinforcement learning for partially observable data poisoning attack in crowdsensing systems," *IEEE Internet of Things Journal*, 2020.

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TNSE.2020.3004312

[19] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963–1971, 2020.
[20] Z. Tian, X. Gao, S. Su, and J. Qiu, "Vcash: A novel reputation framework for identifying denial of traffic service in internet of connected vehicles," *IEEE Internet of Things Journal*, 2020.
[21] Z. Tian, W. Shi, Y. Wang, C. Zhu, X. Du, S. Su, Y. Sun, and N. Guizani, "Real-time lateral movement detection based on evidence reasoning network for edge computing environment," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4285–4294, 2019.
[22] Z. Tian, S. Su, W. Shi, X. Du, M. Guizani, and X. Yu, "A data-driven method for future internet route decision modeling," *Future Generation Computer Systems*, vol. 95, pp. 212–220, 2019.
[23] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, (Madrid, Spain), 2007.
[24] S. J. Yen and Y. S. Lee, "Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset," *Intelligent Control and Automation*, vol. 344, pp. 731–740, 2006.
[25] L. Bao, C. Juan, J. Li, and Y. Zhang, "Boosted near-miss under-sampling on svm ensembles for concept detection in large-scale imbalanced datasets," *Neurocomputing*, vol. 172, pp. 198–206, 2016.
[26] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.
[27] R. Blagus and L. Lusa, "Smote for high-dimensional class-imbalanced data," *Blagus and Lusa BMC Bioinformatics*, vol. 14, p. 106, 2013.
[28] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA2009)*, (Ottawa, Canada), 2009.
[29] R. Sutten and G. Andrew, *Reinforcement Learning: An Introduction*. 2014-2015.
[30] C. Watkins, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.
[31] J. Valenzuela, J. Wang, and N. Bissinger, "Real-time intrusion detection in power system operations," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1052–1062, 2019.
[32] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
[33] S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *IntelliSys 2016: Proceedings of SAI Intelligent Systems Conference (IntelliSys)*, vol. 16, (Springer, Cham), pp. 426–440, 2017.
[34] L. Dhanabal and S. P. Shantharajah, "A study on nsl-kdd dataset for intrusion detection system based on classification algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, 2015.
[35] T. Lai, "Stochastic approximation," *The Annals of Statistics*, vol. 31, no. 2, pp. 391–406, 2003.
[36] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–306, 2014.

Copyright (c) 2021 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.