Expert Systems with Applications, Volume 141, 2020, 112963. https://doi.org/10.1016/j.eswa.2019.112963

Application of deep reinforcement learning to intrusion detection for supervised problems

Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas

Dpto. TSyCeIT, ETSIT, Universidad de Valladolid, Paseo de Belén 15, Valladolid 47011, Spain; (mlopezm@ieee.org; belcar@tel.uva.es; antoniojavier.sanchez@uva.es)
Correspondence: mlopezm@ieee.org; Tel.: +34-983-423-980, Fax: +34-983-423-667

The authors declare that there is no conflict of interest regarding the publication of this paper

Abstract -- The application of new techniques to increase the performance of intrusion detection systems is crucial in modern data networks with a growing threat of cyber-attacks. These attacks impose a greater risk on network services that are increasingly important from a social end economical point of view. In this work we present a novel application of several deep reinforcement learning (DRL) algorithms to intrusion detection using a labeled dataset. We present how to perform supervised learning based on a DRL framework.

The implementation of a reward function aligned with the detection of intrusions is extremely difficult for Intrusion Detection Systems (IDS) since there is no automatic way to identify intrusions. Usually the identification is performed manually and stored in datasets of network features associated with intrusion events. These datasets are used to train supervised machine learning algorithms for classifying intrusion events. In this paper we apply DRL using two of these datasets: NSL-KDD and AWID datasets. As a novel approach, we have made a conceptual modification of the classic DRL paradigm (based on interaction with a live environment), replacing the environment with a sampling function of recorded training intrusions. This new pseudo-environment, in addition to sampling the training dataset, generates rewards based on detection errors found during training.

We present the results of applying our technique to four of the most relevant DRL models: Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Policy Gradient (PG) and Actor-Critic (AC). The best results are obtained for the DDQN algorithm.

We show that DRL, with our model and some parameter adjustments, can improve the results of intrusion detection in comparison with existing machine learning and deep learning techniques. Besides, the classifier obtained with DRL is faster than alternative models. A comprehensive comparison of the results obtained with other machine learning models is provided for the AWID and NSL-KDD datasets, together with the lessons learned from the application of several design alternatives to the four DRL models.

Index Terms—intrusion detection; data networks; reinforcement learning; deep learning

1. Introduction

Considering the importance of current security attacks on networks, the economic importance of services running on these networks (e.g. IoT) and the increased demands on data networks imposed by these services, it is more important than ever to rely on new algorithms capable of detecting intrusions in a fast and reliable manner. It is possible to classify these algorithms by detection approach in signature-based detection and anomaly-based detection (Bhuyan, Bhattacharyya, & Kalita, 2014). Signature-based detection uses a database of previously identified bad patterns to report an attack; while anomaly-based detection uses a machine learning model to classify traffic as good or bad, based in a training dataset of network features and associated intrusion labels.

For this work, we have applied anomaly-based supervised machine learning (ML) models to two different, and, well-known, intrusion detection datasets: the NSL-KDD dataset (Tavallaee et al., 2009). and the AWID dataset (Kolias et al., 2016).

A supervised ML model applies a generic learning algorithm that tunes a number of parameters to optimize prediction. At the center of this learning process is a dataset of recorded samples. These samples are formed by a vector of features (network features in our case) with associated pre-assigned labels. These labels are the elements that the algorithm must predict for a new vector of features.

Contrary to the supervised learning framework, in a classic DRL framework (Arulkumaran et al., 2017) there are two basic components: an agent and the environment. The agent can observe the state of the environment and produce an action. The action is received by the environment generating a new state and a reward, which is associated to the good or bad result of the action. The objective of the learning framework is that the agent can learn to deliver an optimized sequence of actions that maximizes the total sum of rewards. In a classic DRL framework there is no stored dataset of features (states) with their associated actions (label). Instead, the framework attempts to learn this correspondence through an intermediate function (the reward function), and learning is not based on instantaneous information about best actions (feature samples and labels in a training dataset) but on a series of indications (rewards) on the value of intermediate actions with the ultimate goal of maximizing the total sum of rewards.

Considering the differences between the supervised ML framework and the DRL framework, our objective for this work is to show that, with some small adaptations, we can apply a DRL algorithm using a dataset of labeled samples without interacting with a live environment, as required by the classic DRL framework. With this aim, we assimilate the actions with the intrusion labels and the states with the network features. The environment is simulated in such a way that it responds with a positive reward in case of positive detection and a negative one in the opposite case. Any new state provided by the environment corresponds to the sampled features of the training dataset. An important result is that the succession of states does not form a sequence since the samples in the dataset are usually independently distributed. This is not the case for classic DRL problems, where the states form a natural sequence and the next state depends on previous states and actions. Nevertheless, the machinery of DRL algorithms can be applied also in this case, but we must take special care with the values of some parameters if we want to obtain good results. Such a parameter is the discount factor (Sutton & Barto, 1998), whose value must be very low, contrary to the value that is generally found in classic DRL configurations. Section 4 shows the great influence of the value of the discount factor on the detection metrics and Section 3.2.2 gives the reason for that impact.

When applying DRL we can use different algorithms based on different problem requirements and assumptions. We have analyzed the adequacy of four DRL algorithms to our problem: Deep Q-Network (DQN) (Mnih et al., 2013), Double Deep Q-Network (DDQN) (Van Hasselt et al., 2015), Policy Gradient (PG) (Bartlett & Baxter, 2011) and Actor-Critic (AC) (Grondman et al., 2012). In section 4 a comprehensive comparison of the results obtained from these models, together with results obtained from other commonly applied machine learning models is provided, considering different common metrics: precision, F1, accuracy and recall. The conclusion of this comparison is that our proposed framework, using the DDQN algorithm, offers a prediction performance better or similar to the state-of-the-art (SOTA) models, for the two datasets, with the additional advantage of being faster at prediction time.

In addition to the novelty, the DRL models for intrusion detection presented here show many advantages over other ML models. The main contributions of this work are to demonstrate these advantages and present them as a good alternative to other ML models. The list of advantages are: (1) the neural networks used to implement the classifier: Policy, Value or Q functions, are simple and fast, which makes them appropriate for new highly demanding networks e.g. Internet of Things (IoT) Networks (Zarpelo et al., 2017); (2) the resulting neural network can be deployed in modern high-performance distributed platforms (e.g. Tensorflow); (3) the reward function used to drive detection can be extremely flexible and it is not required to be differentiable; (4) the nature of the model allows a simple update of the parameters learned in case of new data available (on-line learning).

The paper is organized as follows: section 2 identifies related works, section 3 describes the work performed and the models analyzed in the paper, section 4 shows the results obtained, and finally section 5 provides discussion and conclusions.

2. Related works

The NSL-KDD and AWID datasets have been widely used in the literature, applying many algorithms. However, the results obtained are difficult to compare, since they depend to a large extent on the test set used. Both datasets provide a standard test set, but many published articles deviate from this, making it difficult to compare the results.

Currently there are no works presenting results of the application of DRL for intrusion detection with the same premises used in this work, but there are many considering existing machine learning and deep learning models. This section presents the most representative of these works applied to the NSL-KDD and AWID datasets. We also discuss the most significant works applying reinforcement learning (RL) to intrusion detection and classification in general.

Machine learning for intrusion detection: There are a number of excellent reviews on the taxonomy, status and current developments of ML for cybersecurity (Hussain et al., 2019; Bhuyan, Bhattacharyya & Kalita, 2014), which is a sub-area of the more generic application of ML to the analysis and prediction of network traffic (Wang et al., 2018; Boutaba et al., 2018).

ML supervised (da Costa et al., 2019) and unsupervised (Nisioti et al., 2018) methods have been widely applied to intrusion detection for data networking, which is currently and area of active research. The classic ML models applied to IDS have been (Tsai et al., 2009; Yavanoglu & Aydos, 2017): Support Vector Machine (SVM), Multilayer Perceptron (MLP), K-Nearest Neighbors (KNN), Decision Trees (DT), Naive Bayes (NB) and Random Forest. It is important to mention the increasing use of the new deep learning models, in particular convolutional and recurrent networks and generative models (Berman et al., 2019; Kim & Aminanto, 2017; Chalapathy & Chawla, 2019)

Machine learning for intrusion detection & NSL-KDD: The NSL-KDD dataset is a classic intrusion detection dataset used in a multitude of previous and recent research works. In Ingre & Yadav (2015) is reported an accuracy of 81.2% for test data and 99.3% for training data with an MLP with three layers. Authors in Ibrahim et al. (2013) apply a Self-Organizing Map (SOM) to obtain a recall of 75.49% on NSL-KDD test data. The work in Panda, Abraham & Patra (2010) employs Naive Bayes with several feature engineering methods, reporting an accuracy of 96.5% with an unclear test set. Similarly, Dhanabal & Shantharajah (2015) achieves an accuracy of 99.1% using several ML models (Support Vector Machine, Naïve Bayes...) with a previously performed dimensionality reduction on the features, but it is not clear the dataset used for testing. Again, in Kamel et al. (2016) they report an accuracy of 99.9% with AdaBoost and a selection of features using a wrapper model; they use a subset of the NSL-KDD dataset for training and an unclear test set. The work in Patil & Pattewar (2014) employs AdaBoost with simple decision stumps as weak learners, reporting sensitivity (recall) of 90% on test data. In Bhuyan, Bhattacharyya & Kalita (2014) is explained why and how the NSL-KDD data set was created and provides prediction results using several ML models with a best reported accuracy of 82.02% with Naive Bayes and the complete NSL_KDD dataset for training and testing. With a scenario of 5-label prediction, in Lopez-Martin, Carro, Sanchez-Esguevillas & Lloret (2017) an accuracy of 80% is obtained for the complete NSL-KDD test data set, using a variational autoencoder. Following similar principles, Chen et al. (2018) presents a convolutional autoencoder for the NSL-KDD with 2-labels attaining an accuracy of 96.87%. In the same line, Shone et al. (2018) presents a novel stacked nonsymmetric deep autoencoder with an overall accuracy (5-labels) of 97.85%. Woo, Song & Choi (2019) applies feature selection and a correct MLP layer configuration to achieve an average accuracy of 98.5% for the NSL-KDD. The excellent review of Berman et al. (2019) presents the application of different configurations of autoencoders, MLPs, recurrent networks and restricted Boltzmann machines to NSL-KDD with different results in prediction accuracy. Thomas & Pavithran (2018) provides a review of 8 works using NSL-KDD with different ML models and data preparation processes, with a maximum accuracy of 99.81% for a hybrid model of J48, Random Tree, REPTree, AdaBoostM1, Decision Stump and NB. In Javaid et al. (2016) self-taught learning (STL) is proposed as a model based on deep learning that achieves a F1 score, for 2 labels, of 90.4%; STL consists of an initial dimensionality reduction model that is trained with labeled and unlabeled data to obtain a good feature representation; this initial model is used later to train a common classifier, using the new features produced by the initial model as input to the classifier. The work in Bhattacharjee, Fujail & Begum (2017) uses two types of clustering algorithms (K-Means and Fuzzy C-Means) for intrusion detection in NSL-KDD (5-labels) with the best attack detection of 45.95% for Fuzzy C-Means.

Machine learning for intrusion detection & AWID: The work that originally presented the AWID dataset (Kolias et al., 2016) provides a comprehensive analysis of the performance of different machine learning algorithms for predicting attacks. The authors of the dataset apply 8 classification algorithms (AdaBoost, Hyperpipes, J48, Naïve Bayes, OneR, Random Forest, Random Tree and ZeroR). They perform an exhaustive comparison of the models, being the model J48 (an implementation of C4.5) who presents the best results: accuracy of 96% and F1of 94.8%. Wang et al. (2019) employs Stacked Autoencoder (SAE) and Deep Neural Networks (DNN) to perform classification of the four types of AWID attacks with accuracies from 99.9% to 73.1%. The review in Berman et al. (2019) presents also and autoencoder applied to AWID. Abdulhammed et al. (2018) applies feature selection and seven classic ML models to AWID, with Random Forest providing the best accuracy results (99.64%). Rezvy et al. (2019) achieves an overall accuracy of 99.9% for AWID with a deep autoencoder. The work of Qin et al. (2018) proposes a fast algorithm based on SVM applied to the AWID dataset with a reduced number of features, obtaining an accuracy between 87.34% and 99.98% for the different attack types. Nivaashini & Thangaraj (2019) presents a review of common ML models applied to intrusion detection for wireless networks; in this study, AWID is introduced as the reference dataset for these networks. In Moshkov (2017) gradient boosting, Random Forest and MLP models are applied to intrusion detection for wireless networks, proposing AWID as the chosen dataset with the best results obtained for gradient boosting. The work in Thanthrige, Samarabandu & Wang (2016) explores the importance of feature selection using information gain and Chi-squared statistics, achieving an improvement of 2.4% after feature reduction using Random Tree with a final overall accuracy of 95.12%.

Reinforcement learning for intrusion detection: The work in Servin (2007) presents an anomaly detector based on reinforcement learning with a simulated network environment, where anomalies are injected in a controlled manner and the reward is based on the correct detection of the anomalies. From a logical point of view, this experiment presents similarities with the present work, since, although the environment is simulated, the reward function is manually controlled, and it is not generated by the environment itself, and the real-time generation of sequences of actions, states and rewards could assimilate to be recorded on a dataset, since there is no sequential information in successive states. However, the techniques applied in Servin (2007) notably differ from the presented work since they use a Q-learning algorithm based on a look-up table that requires a discretization of states to avoid an explosion in the table size. Instead, current DRL models use function approximators based on neural networks (NN), which allow generalizing to states of any size with continuous or discrete values.

In Xu (2010); Sukhanov, Kovalev & Stýskala (2015) and Xu & Xie (2005) they also employ look-up tables with temporal difference (TD) learning (Sutton, 1988) for intrusion detection in live sequences of traffic flows. Similarly, Xu (2006) applies a kernel version of TD to a Markov chain prediction problem and Xu & Luo (2007) applies the same model to a host-based IDS. Authors of Servin (2009) and Malialis (2014), in a different framework, perform intrusion detection by applying reinforcement learning based on a multi-agent architecture forming a hierarchy to detect anomalies, while in our work we use a single agent. They also require the discretization of the state space to apply a look-up table. None of the above works apply DRL models to intrusion detection.

Deokar & Hazarnis (2012) proposes a framework for using log-files at different system levels (e.g. client, proxy, firewall, network and system) based on a hierarchy of log correlations implemented with association rules (AR), where the strength of each AR is controlled by a reinforcement learning model. The framework assumes interaction with a live environment.

It is worth mentioning the interest of reinforcement learning models applied to cyber-physical-systems (CPS) (i.e. evolution of embedded systems with highly integrated physical, computer-based and network elements). In this field, Feng & Xu (2017) presents an application of DRL (actor critic) to a CPS system with a dynamical system perspective where the objective is to defend the system against cyber-attacks. In the same area, Akazaki et al. (2018) introduces a study to prevent the falsification of entries to a CPS using DRL with the intention of generating 'counterexamples' that could endanger the system. The detection of these 'counterexamples' is critical to protect a system against possible attacks, in the same line presented in the study of Goodfellow I., Shlens J. & Szegedy Ch. (2015) to guarantee the robustness in the classification of images.

Nguyen & Reddi (2019) presents a complete and updated review of DRL for Cybersecurity. The works presented are based on a live environment (real or simulated).

Reinforcement learning for classification: In Wiering et al. (2011) is presented a DRL approach based in an actor-critic model to perform classification over several well-known UCI datasets (e.g. Iris, Hepatitis...). Considering the technique applied, this work can be considered similar to ours; however, the approach taken to perform classification is based on extending the state space dimension with additional ancillary variables (features) which are used as memory cells in a copy and erase process together with a complex reward scheme. This has a major impact on complexity and computational performance (e.g. training times), and it is a very different strategy from the one presented in this work, which applies very simple reward functions and does not require feature engineering.

3. Work description

In this Section we describe the datasets chosen for the experiments and the different DRL models employed. The datasets are described in Section 3.1. The proposed algorithms, with our framework adaptations, are presented in detail in Section 3.2.

3.1. Intrusion detection datasets

For this work, we have considered two datasets that satisfy a series of requirements: (1) labeled datasets, (2) unbalanced but with a different level of imbalance, which allows studying the behavior in different conditions, (3) a predefined split for the training and test datasets, which provides a means to compare results from different works, (4) well-known datasets, which make available a sufficient number of results from previous works, (5) to include older and more recent datasets, to increase generality/variability, (6) data coming from different network architectures (e.g. fixed-line vs wireless networks), and (7) the need to have a data volume large enough to have significant results, but limited by practical restrictions of memory and CPU time. Considering these aspects, we have opted for the NSL-KDD and AWID datasets, which satisfy most of the listed requirements (Ring et al., 2019).

In addition, NSL-KDD and AWID are among the most frequently used intrusion detection datasets, in two recent literature reviews: Ring et al. (2019) and da Costa et al. (2019). This importance can also be inferred from the significant number of current works related to both datasets; as a summary: Shone et al. (2018), Berman et al. (2019), Javaid et al. (2016), Nivaashini & Thangaraj (2019), Chen et al. (2018), Thanthrige, Samarabandu & Wang (2016), Moshkov (2017), Rezvy et al. (2019), Qin et al. (2018), Abdulhammed et al. (2018), Thomas & Pavithran (2018), Bhattacharjee, Fujail & Begum (2017), Yavanoglu & Aydos (2017), Woo, Song & Choi (2019).

3.1.1 NSL-KDD

We have used the well-known IDS dataset: NSL-KDD. It is an evolution of the original KDD-99 (Tavallaee et al., 2009) dataset, which has the problem of being biased towards the more frequent labels. For this reason, the detection scores on the KDD99 are usually higher than for the NSL-KDD dataset. We consider that the results from NSL-KDD are more useful and realistic than those obtained from KDD-99, and, despite being a classic dataset; we are inclined to use it since it is a well-known dataset, where comparison of results with other models is feasible.

The NSL-KDD dataset has 125973 training samples and 22544 test samples, with 41 features: 38 continuous and 3 categorical (discrete valued). We have performed a data preparation consisting of: scaling the continuous features to the range [0–1] and one-hot encoding the categorical features. This provides a final dataset with 122 features: 38 continuous and 84 with binary values ({0, 1}) associated to the three categorical features (one-hot encoded). NSL-KDD is an unbalanced dataset with a frequency of 43.1% and 1.7% for the most and least frequent labels.

The training dataset has 23 possible labels (normal plus 22 labels associated to different types of anomaly). However, the test dataset has 38 label values, implying that the test data has anomalies not presented at training time. Up to 16,6% of the samples in the test dataset correspond to labels unique to the test dataset, and not used for training. The existence of new labels at testing introduces an additional challenge to the algorithms.

As presented in Tavallaee et al. (2009), the training/test labels can be additionally grouped into two significant categories: normal and anomaly. This final grouping makes sense, since with the NSL-KDD dataset we want to verify the capacity of the classifier to handle a different distribution of attacks between the training and test dataset. While the AWID dataset has been used to verify the ability to handle a very unbalanced dataset.

In Fig.1 is presented the frequency of these aggregated categories in the NSL-KDD training and test datasets.

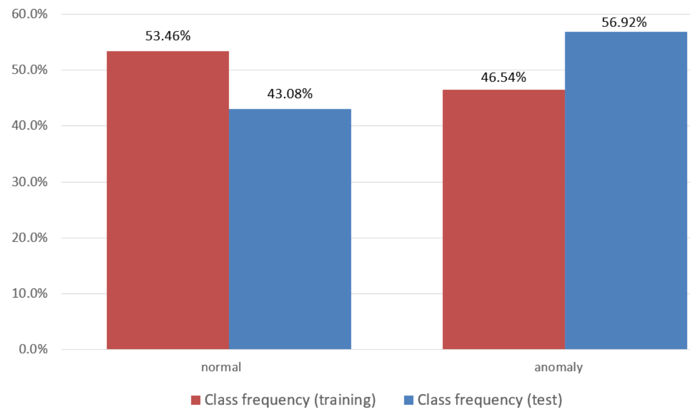


Fig. 1. Frequency distribution of intrusions for the training and test datasets (NSL-KDD).

In this study, we apply all models to the NSL-KDD dataset, obtaining the most relevant performance metrics: accuracy, precision, recall and F1, for the detection of two label values: normal and anomaly. We provide all results using the training dataset of 125973 samples and the full test dataset of 22544 samples.

3.1.2 AWID

Aegean Wi-Fi Intrusion Dataset (AWID) (Kolias et al., 2016) is a public dataset containing normal traffic along with three types of attacks against IEEE 802.11 networks. AWID is a larger and more recent dataset than NSL-KDD.

From the diverse groups of data provided by AWID, we have chosen the AWID-CLS-R dataset. This dataset provides a separated training and test datasets. It provides 4 labels to classify: normal, flooding, injection and impersonation. The dataset contains 154 features (continuous and categorical) with 1,795,574 and 575,642 samples for the training and test datasets, respectively. The number of features can be reduced to 24 by eliminating the features with null and constant values, and the features that contain network addresses which are sample specific and cannot be generalized to new test data. Continuous features have been scaled to the [0–1] range and all categorical features are one-hot encoded.

This is an extremely unbalanced dataset with 91% of normal samples and 9% associated with anomalies: 2.7% flooding, 2.7% impersonation and 3.6% injection. Fig. 2 shows the class distribution of the training and test AWID datasets. This dataset is even more unbalanced than NSL-KDD, but, unlike NSL-KDD, the label distribution for the training and test datasets are quite similar. Therefore, the interest of experimenting together with the AWID and NSL-KDD datasets is that each offers different challenges to a classification algorithm.

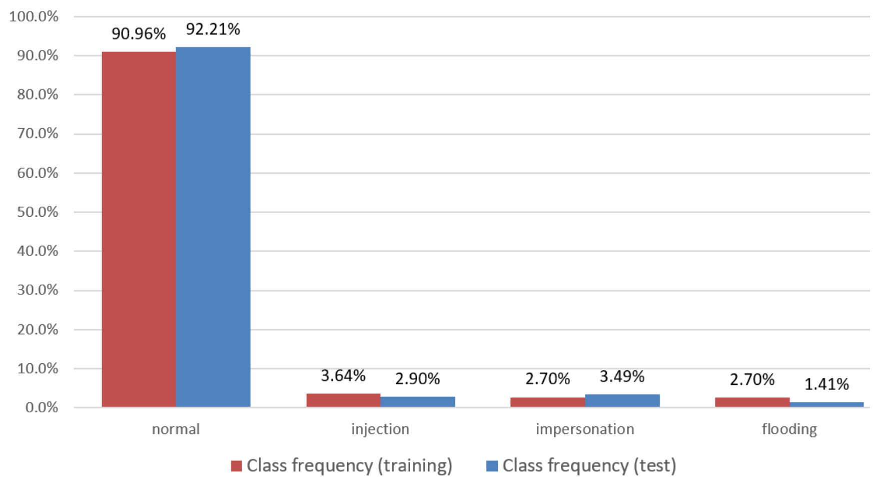


Fig. 2. Frequency distribution of intrusion classes for the training and test datasets (AWID).

3.2. Models description

This section explains the different DRL models studied in this work. There are excellent introductions to DRL (Arulkumaran et al., 2017), here we provide a brief summary. DRL is a type of RL which uses deep learning models (e.g. NN, convolutional NN…) as function approximators for the policy and/or value functions used in RL.

RL is based in the Markov Decision Process (MDP) theory. An MDP is a tuple $\langle S, A, T, R\rangle$, where S is a set of states, A is a set of actions, T is a mapping defining the transition probabilities from every state-action pair to every possible new state, and R is a reward function which associates a real value (reward) to every state-action pair. In an MDP the function T follows the Markov property, which means that the transition probability to a new state depends exclusively on the current state and action regardless of previous history. Once the MDP is defined, a policy for an MDP is a mapping from each state to an action. The objective of an MDP is to learn the optimal policy that gives the best action for every state in order to achieve a best sum of expected rewards. This optimality criterion can be further selected as a simple sum of rewards, an average or a sum of discounted rewards (when the current rewards are considered more important than future ones).

An MDP is the theoretical framework used to characterize an agent interacting with an environment in a sequential decision-making process; where the environment implements the T and R functions and the agent implements the policy. The interaction between the agent and the environment is usually discretized in a sequence of "time-steps", in which the agent provides a new action to the environment that in turn generates a state transition and a possible new reward.

The link between the optimality criterion and the policy is usually made by defining a value function, which is an estimate of the value associated with each state. That is, an estimate of how good it is to be in a certain state considering that we will move forward with the current policy. The value function can be a V-function or a Q-function. The V-function estimate a value for each state and the Q-function estimate a value for each state-action pair. Both are related, being the value of the Q-function the sum of the reward for the given state-action pair plus the value of the V-function for the next state generated by the environment.

When the T and R functions of the MDP are known, we can use model-based solution techniques to obtain the optimal policy. These techniques are based on the previously defined value function plus some theoretical results: Bellman optimality equation and Generalized Policy Iteration (GPI) (Sutton & Barto, 1998). These methods are usually considered in the field of dynamic programming solutions, with two alternative implementations: policy and value iteration; the difference between them being the way to carry out the GPI mechanism. Even when the aforementioned theory and implementation alternatives are applicable to model-based solutions, they form the basis of all RL methods (model-based or model-free).

When T and R are not known, we must apply model-free solution techniques to obtain the optimal policy. In this case, there are also two options, we can try first to learn the model (T and R) and then apply the previous techniques once T and R are known, and, the other alternative is to try to learn the best policy directly without knowing first T and R. This latter scenario is the one taken by most of the RL models, and it is the one considered for this work. The problem in this scenario is that not knowing the dynamics (T and R) of the environment prevents us from using the Bellman equation, which requires knowing the transition probabilities of each state to all other possible states. There are different solutions in this case: (1) TD learning algorithms, (2) policy gradient and (3) Monte Carlo methods. TD methods are based on considering a single transition of states (from the current state to the next state under the current policy) in a slightly modified Bellman equation, instead of considering all possible transitions as required by the original Bellman equation. DQN, DDQN and actor-critic methods are based on TD learning. Policy gradient methods try to learn a policy function directly using gradient descent to optimize the expected sum of discounted rewards under a current policy. Monte Carlo methods are based in exploring the frequencies of state and action pairs and their associated sum of rewards along sampled trajectories, for example, evaluating a value function for one state will be based on the average value obtained from several trajectories starting from that state.

To achieve an optimal policy, it is important to sweep (explore) as much as possible the state-action space. The main exploration approaches are ε-greedy and action-probability based. In ε-greedy the best action is selected with probability p or a random action with probability 1-p. The exploration based in action-probability assumes that the policy (directly or indirectly) provides a probability for each of the actions, which allows a sampling process according to this probability distribution.

We can conclude that the learning of the value or policy functions is based on different types of iterative processes, where each iteration generates a function adjustment. To implement these functions there are two main alternatives: look-up tables and function approximators. A look-up table is based on a data structure that stores the value of the function for each possible combination of input values (state and/or action); meanwhile, a function approximator calculates a value for each possible combination. The chosen alternative is very important when the state and/or action spaces have a high dimensionality, with a strong impact on performance and storage capacity. All DRL methods use function approximators based on different NN variants.

3.2.1 Datasets preparation

A generic task for all models is to handle the dataset to create the mini-batches (set of samples used in a training iteration) that each specific model will use. The training dataset contains N samples of network features and associated intrusion labels with several possible values (binary or multiclass anomaly). To assimilate these elements to DRL concepts, we consider the network features as states and the label values as actions. The training is performed with mini-batches of samples formed by: (1) a state, (2) its correct label and (3) a next state.

A mini-batch will be a subset of samples drawn at random from the dataset. Each training pass is done with a different mini-batch that is updated by random sampling from the dataset.

Fig. 3 shows the structure of a mini-batch used by the DQN, DDQN and actor-critic models. In this case, a mini-batch is formed by n+1 random samples of the dataset. The generation process for each mini-batch is to randomly permute the dataset before the process is initiated, and then choose n+1 consecutive samples starting from a random index (t).

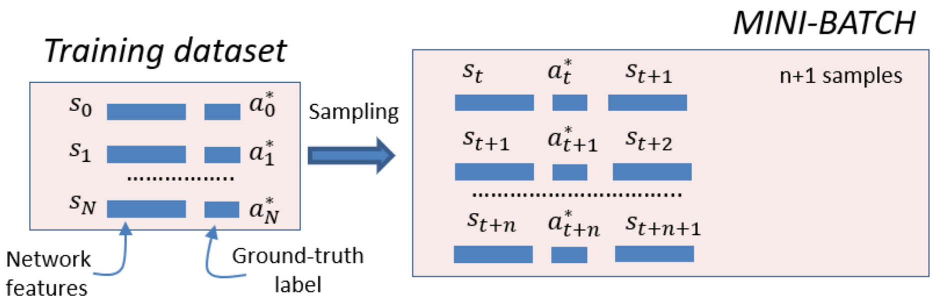
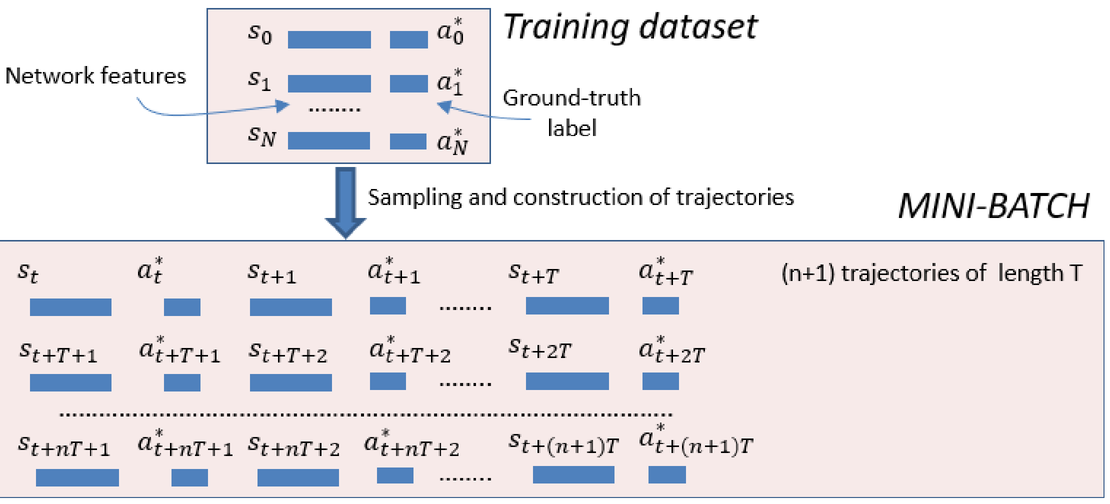


Fig. 3. Dataset preparation for the training of the DQN, DDQN and actor-critic models

The policy gradient model forms the mini-batches in a different way. Policy gradient (Bartlett & Baxter, 2011) needs a sequence of states forming an episode (trajectory), which is the reason to build training mini-batches integrated by n trajectories of length T. Contrary to what happens when dealing with a live environment where the length of the trajectories is not fixed, in our case we can choose the length of the trajectories since we construct them by sampling from the dataset.

In Fig. 4 is presented the structure of a mini-batch used by the policy gradient model. The structure is formed in a similar way to that presented in Fig 3, but, in this case, we create n+1 consecutive trajectories of length T. To ensure that the trajectories are randomly chosen, we also make a random permutation of the dataset before the construction of each mini-batch.



Training dataset
s_0 a_0^*
s_1 a_1^*
s_N a_N^*
Network features
Ground-truth label
Sampling and construction of trajectories
MINI-BATCH
s_t a_t^* s_{t+1} a_{t+1}^* s_{t+T} a_{t+T}^*
(n+1) trajectories of length T
s_{t+T+1} a_{t+T+1}^* s_{t+T+2} a_{t+T+2}^* s_{t+2T} a_{t+2T}^*
s_{t+nT+1} a_{t+nT+1}^* s_{t+nT+2} a_{t+nT+2}^* $s_{t+(n+1)T}$ $a_{t+(n+1)T}^*$

Fig. 4. Dataset preparation for the training of the policy gradient model

The dataset preparation depicted in Fig 3 and 4 will apply to the two datasets used in this work (NSL-KDD and AWID)

3.2.2 Models detail

This section presents in detail the different DRL models used for this work.

3.2.2.1 DQN

The algorithm employed for training DQN (Mnih et al., 2013) is depicted in Fig. 5.

The objective of the DQN model is to approximate the Q function. A Q function provides the maximum expected reward under a specific state and action, therefore, depending on a state and action pair: $Q(s, a)$. Once the Q function is obtained, then we can easily get the policy function which is the function that designates the action to take for each state. The policy function depends on the state and it is derived from the Q function as follows: Policy $(s) = arg_a \max(Q(s, a))$. That is, it is the action that maximizes the Q-value.

The algorithm depicted in Fig 5 starts from a generic sample formed by the current state (s_t), the ground-truth label for the current state (a_t^*) and the next state (s_{t+1}).This generic sample is part of a mini-batch of n samples. The process followed to create this generic sample from the original dataset is presented in section 3.2.1. Each training iteration of the algorithm will process all the samples of a mini-batch, and for each iteration a new mini-batch is built following the process shown in section 3.2.1.

A neural network (NN) is used as a function approximator for the Q function. We have used a simple NN of 3-layers, with ReLU activation for all layers, including the last one to ensure a positive Q-value. The NN training is done with a Mean Square Error loss between the Q-value estimated by the NN for the current state ($\widehat{q}_t$) and a reference value: q_{ref}, obtained by adding the current reward (r_t) to the Q-value for the next state ($\widehat{q}_{t+1}$) multiplied by a discount factor (λ).

The reward is a 1/0 reward associated, respectively, to a correct/incorrect prediction. In Fig. 5, the ground truth label value for the current state is represented by a_t^* and the predicted value by $\hat{a}_t$. When these two values are equal the reward is 1 and 0 otherwise.

To obtain the predicted value for the current state ($\hat{a}_t$) we iterate the Q function with the current state (s_t) and all possible values of the label ($\{a\}$). This iteration is represented in Fig. 5 as: $Q(s_t, \{a\})$, which is a vector of values and is shown as a thicker arrow: $Q(s_t, \{a\}) = \left[Q(s_t, \{a\}_0), Q(s_t, \{a\}_1), .. Q\left(s_t, \{a\}_p\right)\right]$, where $\{a\}$ is the set of all possible actions and p is the cardinality of this set..

Then, we choose the action value which produces the maximum Q-value obtained from this iteration: $arg_a\, max(Q(s_t, \{a\}))$. The action chosen is further applied to an ε-greedy algorithm which selects that value with probability p or a random action with probability 1-p. The result of this last step provides the predicted action ($\hat{a}_t$).

The predicted action for the next state ($\hat{a}_{t+1}$) is obtained in a similar way but ignoring the ε-greedy selection (right part in Fig. 5). This predicted action together with the next state (s_{t+1}) is used to obtain the Q-value for the next state ($\hat{q}_{t+1}$) which is used to calculate qref (right part in Fig. 5). The prediction of the Q-value for the next state ($\hat{q}_{t+1}$), as presented on the right part of Fig. 5, can be simplified to: $\hat{q}_{t+1} = \max_a Q(s_{t+1}, \{a\})$.

The best performance was obtained by applying a small value for the discount factor (λ). Small values of λ give more importance to learning the current reward, regardless of the succession of future rewards. This makes sense, considering that (1) the next state is uncorrelated with the present state, and (2) we are actually implementing a classifier disguised as a DRL algorithm, being the real goal to make a correct prediction for the current state (current reward).

In order to make a faster algorithm, our final model used for DQN was slightly modified with respect to the one shown in Fig. 5. The network implementation for the Q function consists of two input vectors (state and action) and one scalar as output (Q-value). This architecture requires iterating the network with all the action values to recognize the value that maximizes the Q function. This iteration slows down the process. As a solution, in Mnih et al. (2013) is presented an alternative network architecture with a single input vector (state) and one output vector with the same length as the number of action values (i.e. number of intrusion classes, since they are one-hot encoded). In this way, each element of the output vector has the Q-value of its associated action value through a single iteration.

In addition to the 1/0 reward, other rewards were tested, in particular, the functions cross-entropy and categorical hinge were used to evaluate the distance between the predicted and ground-truth actions (labels). The simplest 1/0 reward was finally selected due to better performance.

Once the training of the model is completed, the NN implementing the Q function is used for prediction. For a particular state, the Q function will provide a Q-value for each of the possible actions for that state. The predicted action is the one that maximizes the Q-value (no ε-greedy applied for prediction).

The model was trained for 10 epochs, where an epoch is a number of iterations enough to cover the complete dataset.

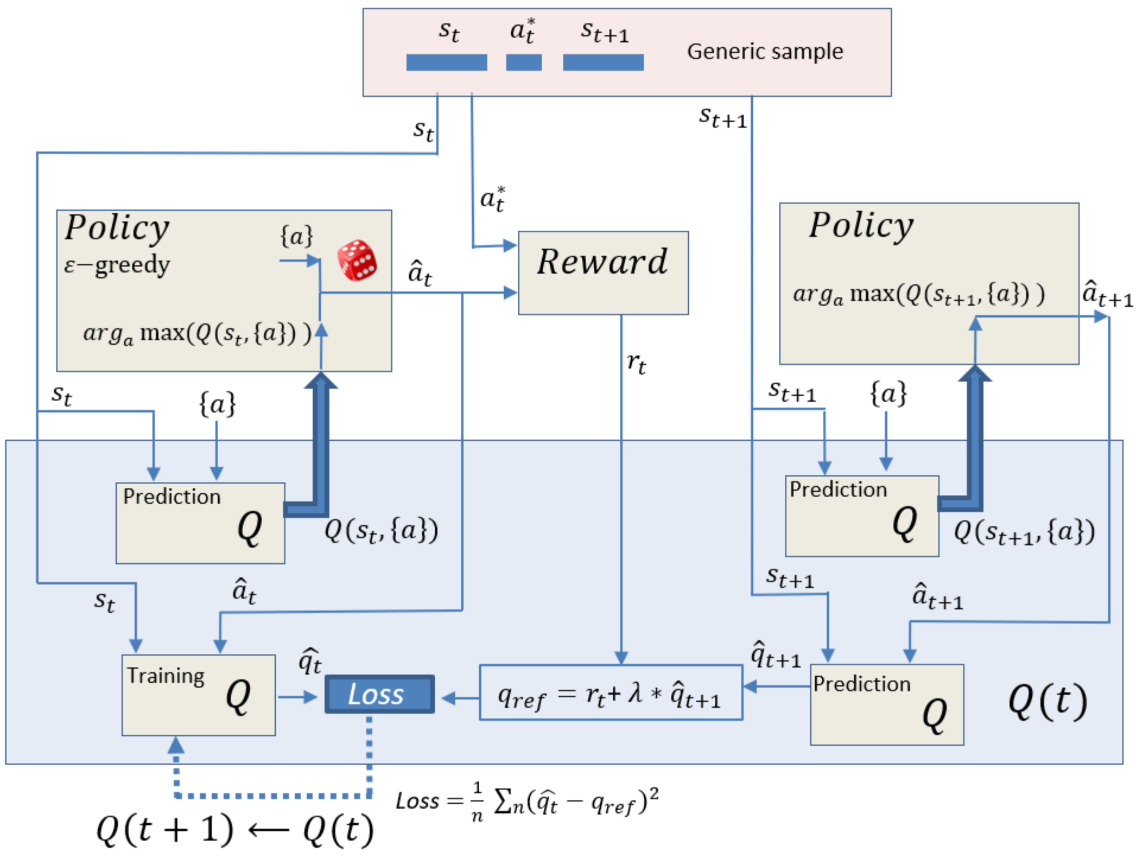


Fig. 5. Schema of the DQN model during training

3.2.2.2 DDQN

The DDQN (Van Hasselt et al., 2015) model is presented in Fig. 6, where the algorithm applied to the training phase is shown.

The DDQN model is very similar to DQN. The only difference being that two NNs are employed in DDQN: one implements a current Q function while the other one implements a target Q function. The target Q function is a copy of the current Q function, but with a delayed synchronization; that is, the copy is made after a certain number of training iterations. The target Q function is used to calculate the Q-value for the next state ($\hat{q}_{t+1}$). The intention of this additional Q function (target Q function) is to avoid the moving target effect when doing gradient descent over $(\widehat{q_t} - q_{ref})^2$, avoiding the recursive dependence of q_{ref} on the training network (Q current) (Van Hasselt et al., 2015).

Besides the addition of the target Q function, the algorithm depicted in Fig. 6 is similar to Fig. 5, and all the mechanisms explained in Section 3.2.2.1 are applicable here. The hyper-parameters used were also similar to the ones for the DQN model.

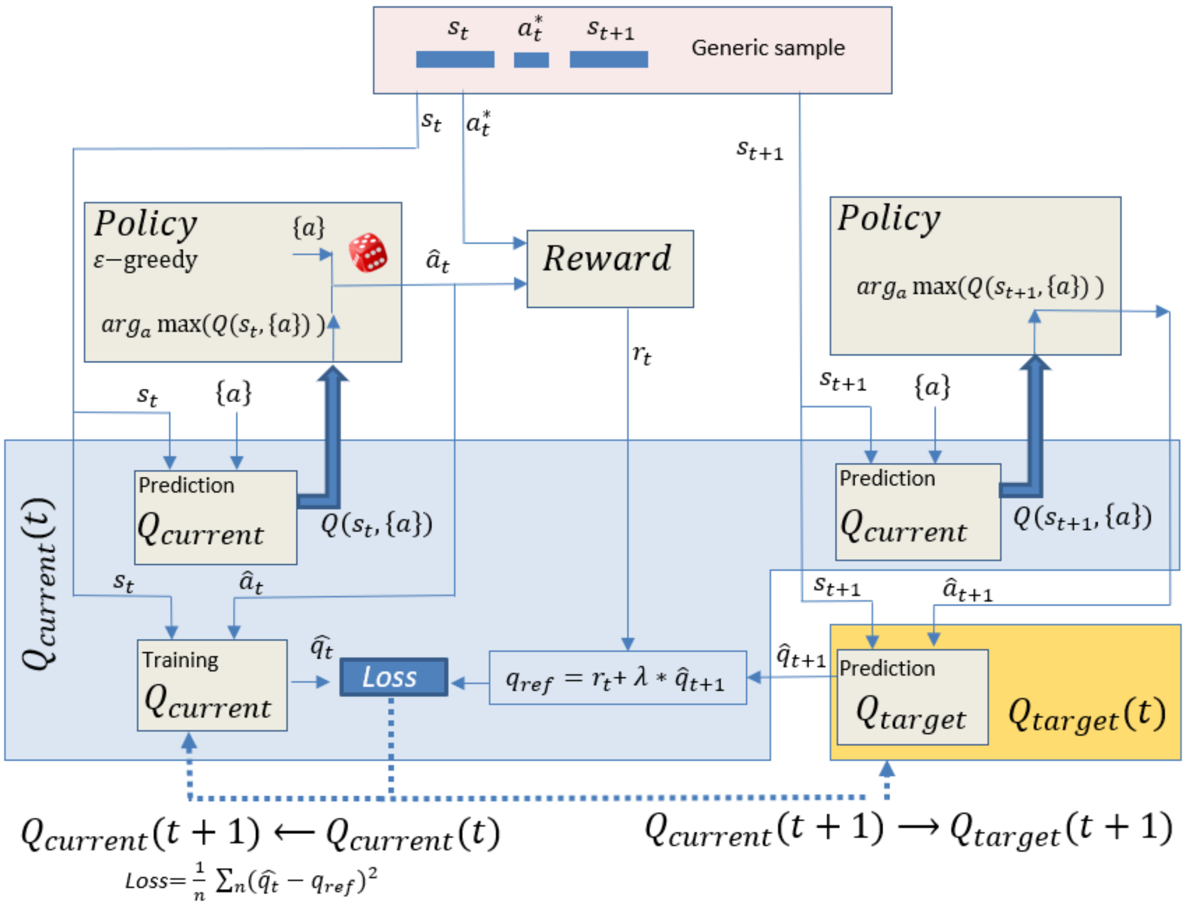

s_t a_t^* s_{t+1} Generic sample
s_t
a_t^*
s_{t+1}
Policy
ε−greedy
{a}
$arg_a \max(Q(s_t,\{a\}))$
$\hat{a}_t$
Reward
r_t
Policy
$arg_a \max(Q(s_{t+1},\{a\}))$
s_t {a}
s_{t+1} {a}
Prediction
$Q_{current}$
$Q(s_t,\{a\})$
Prediction
$Q_{current}$
$Q(s_{t+1},\{a\})$
$Q_{current}(t)$
s_t $\hat{a}_t$
s_{t+1} $\hat{a}_{t+1}$
Training
$Q_{current}$
$\widehat{q_t}$
Loss
$q_{ref} = r_t + \lambda * \hat{q}_{t+1}$
$\hat{q}_{t+1}$
Prediction
Q_{target}
$Q_{target}(t)$
$Q_{current}(t+1) \longleftarrow Q_{current}(t)$
$Loss = \frac{1}{n}\sum_n(\widehat{q_t} - q_{ref})^2$
$Q_{current}(t+1) \longrightarrow Q_{target}(t+1)$

Fig. 6. Schema of the DDQN model during training

3.2.2.3 Policy gradient

The diagram in Fig. 7 shows the details of the training process of the policy gradient model (Bartlett & Baxter, 2011) employed for this work.

Policy gradient is based on training a policy function, which designates the action that must be taken for each possible state. The policy function is approximated by a simple NN that has been implemented with a few layers and ReLU activation for all layers except the last layer which has a softmax activation that provides a probability distribution of the actions ($\pi(a)$).

The algorithm depicted in Fig 7 uses a generic trajectory consisting of a sequence of T pairs formed by a state (s_t), and its associated ground-truth label (a_t^*). This generic trajectory is part of a mini-batch of n trajectories. The process followed to create this generic trajectory from the original dataset is presented in section 3.2.1. Each training iteration of the algorithm will process all the trajectories of a mini-batch, and for each iteration a new mini-batch is built following the process shown in section 3.2.1.

The algorithm begins by predicting the actions using the states and the policy function. The action prediction is done for all the states in a trajectory ($s_{\{T\}}$), producing a sequence of predicted actions: $\hat{a}_{\{T\}}$. These predicted actions are obtained by sampling on the probability distribution of the actions ($\pi\left(a_{\{T\}}\right)$) provided by the policy function. This is represented as the "Prob. Distribution Sampler" in Fig. 7.

We use the symbol $\{T\}$ to represent a sequence along the time-steps of one trajectory. When we use this symbol we can have a sequence of scalar values, as in $r_{\{T\}}$ or a sequence of vectors such as $\pi\left(a_{\{T\}}\right)$ or $\hat{a}_{\{T\}}$, since in the latter case, $\pi(a)$ is a vector of probabilities for each possible action under the current policy, and $\hat{a}_t$ is a one-hot encoded vector with a 1 assigned to the picked action, therefore their extension to a sequence produces a sequence of vectors.

Similarly to Section 3.2.2.1, the reward function generates a 1/0 reward, but, in this case, it is applied to a whole sequence of predicted actions ($\hat{a}_{\{T\}}$) and ground-truth actions ($a_{\{T\}}^*$) in a trajectory. The reward sequence ($r_{\{T\}}$) obtained is transformed into a vector of sums of discounted rewards ($R_{\{T\}}$).

$R_{\{T\}}$ is calculated with the following expression:

$$R_{\{T\}} = \left[\sum_{i=0}^{T} \lambda^i r_{t+i}, \sum_{i=1}^{T} \lambda^i r_{t+i}, \ldots, \sum_{i=T}^{T} \lambda^i r_{t+i}\right] = \left[\sum_{i=0}^{T} \lambda^i r_{t+i}, \sum_{i=1}^{T} \lambda^i r_{t+i}, \ldots, \lambda^T r_{t+T}\right]$$

That is, each term of $R_{\{T\}}$ corresponds to a decreasing sum of terms of consecutive discounted rewards.

From the vector of sums of discounted rewards ($R_{\{T\}}$) we subtract the average of discounted rewards along trajectories ($b_{\{T\}}$), resulting in the vector of advantage values ($A_{\{T\}}$). The vector $b_{\{T\}}$ is also called a baseline. The advantage values provide an estimate on how much the expected return for a certain element of the trajectory (s_t) is better than the average expected return (that is the reason to subtract the baseline from $R_{\{T\}}$).

The scalar product between the sequence of vectors $\pi\left(a_{\{T\}}\right)$ and $\hat{a}_{\{T\}}$, extracts the probability of the picked action for each time-step ($\pi\left(\hat{a}_{\{T\}}\right)$), since $\hat{a}_t$ is a one-hot encoded vector. A product symbol inside a circle is used in Fig. 7 to represent the scalar product.

The loss used to train the NN that approximates the policy function, is a kind of log-loss function formed by the sum along trajectories of the log of the probability of the picked action for a certain element of the trajectory ($\log \pi\left([\hat{a}_{\{T\}}]_i\right)$) multiplied by its corresponding advantage value ($[A_{\{T\}}]_i$). The complete process is depicted in Fig. 7.

When the training is finished, for prediction we use the NN implementing the policy function. For a particular state, the policy function will provide a distribution of probabilities for the actions. In this prediction case, we just choose the action with the highest probability (without sampling).

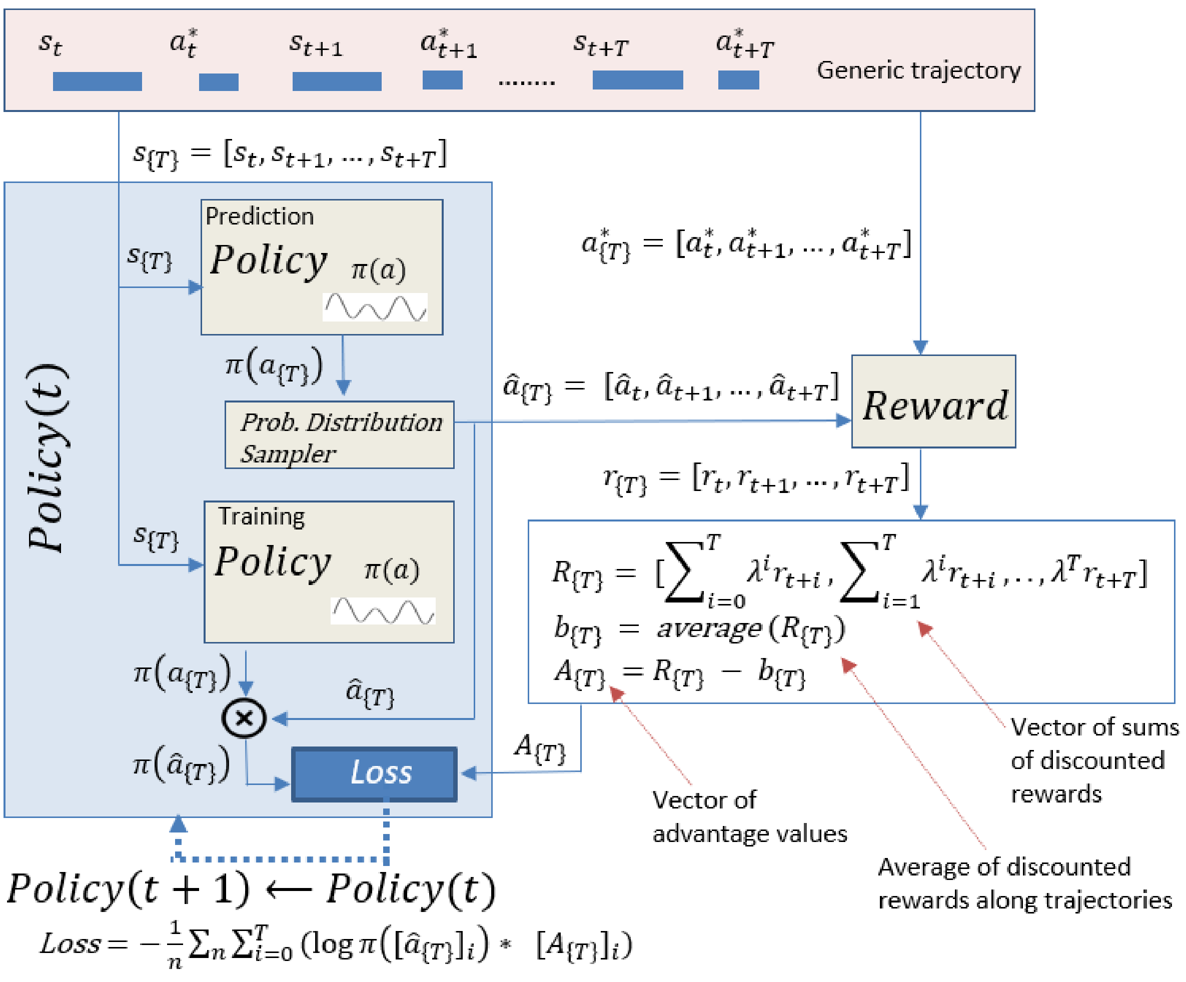

s_t
a_t^*
s_{t+1}
a_{t+1}^*
....
s_{t+T}
a_{t+T}^*
Generic trajectory
$s_{\{T\}} = [s_t, s_{t+1}, \ldots, s_{t+T}]$
$a_{\{T\}}^* = [a_t^*, a_{t+1}^*, \ldots, a_{t+T}^*]$
$Policy(t)$
$s_{\{T\}}$
Prediction
$Policy$ $\pi(a)$
$\pi(a_{\{T\}})$
Prob. Distribution Sampler
$\hat{a}_{\{T\}} = [\hat{a}_t, \hat{a}_{t+1}, \ldots, \hat{a}_{t+T}]$
Reward
$r_{\{T\}} = [r_t, r_{t+1}, \ldots, r_{t+T}]$
$s_{\{T\}}$
Training
$Policy$ $\pi(a)$
$R_{\{T\}} = [\sum_{i=0}^{T} \lambda^i r_{t+i}, \sum_{i=1}^{T} \lambda^i r_{t+i}, \ldots, \lambda^T r_{t+T}]$
$b_{\{T\}} = average(R_{\{T\}})$
$A_{\{T\}} = R_{\{T\}} - b_{\{T\}}$
$\pi(a_{\{T\}})$
$\hat{a}_{\{T\}}$
$\pi(\hat{a}_{\{T\}})$
Loss
$A_{\{T\}}$
Vector of sums of discounted rewards
Vector of advantage values
Average of discounted rewards along trajectories
$Policy(t+1) \leftarrow Policy(t)$
$Loss = -\frac{1}{n} \sum_n \sum_{i=0}^{T} (\log \pi([\hat{a}_{\{T\}}]_i) * [A_{\{T\}}]_i)$

Fig. 7. Schema of the policy gradient model during training

3.2.2.4 Actor-critic

The schema for the training of the actor-critic (Grondman et al., 2012) model is shown in Fig. 8. For the actor-critic model we use two function approximators: one for the policy function, another one for the Value function. The Value function provides the expected sum of rewards starting from a designated state, and the policy function indicates the action to take under a given state. The two function approximators are implemented with two NNs of 3-layers and ReLU activation for all layers except the last layer of the policy function with softmax activation. The softmax activation provides the probability distribution of the actions.

By addressing both functions simultaneously, they both help each other in the training process. Under a given current state (s_t), the policy function is used to estimate the ground truth action (a_t^*). The policy function provides the probability distribution ($\pi(a)$) of the actions under a given state. This probability distribution is used in a sampling process to choose the preferred action ($\hat{a}_t$). The selected action ($\hat{a}_t$) together with the ground-truth action (a_t^*) is used by the reward function to provide a 1/0 reward associated to a correct/incorrect action selection. If the picked action matches the ground-truth label, a reward of 1 is given, otherwise the reward is 0. This is the current reward (r_t).

The algorithm depicted in Fig 8 starts from a generic sample formed by the current state (s_t), the ground-truth label for the current state (a_t^*) and the next state(s_{t+1}).This generic sample is part of a mini-batch of n samples. The process followed to create this generic sample from the original dataset is presented in section 3.2.1.

The previously mentioned current reward (r_t) is used to make an estimation of the value function (R_t)

associated with the current state. This estimation is employed to train the value function (with a mean square error loss). The difference between this estimation of the value function (R_t) and the actual value function (V_t), obtained from the NN approximator, is an estimation of the advantage value (A_t) for the current state. This advantage value is used, similarly to policy gradient, to train the policy function. The policy function is trained with a loss formed by the log of the probability of the picked action ($\hat{a}_t$) under the probability distribution given by the policy function ($\log \pi(\hat{a}_t)$), and weighted by the corresponding advantage value (A_t).

Similar to policy gradient (Section 3.2.2.3) we use a scalar product between the selected action ($\hat{a}_t$) and the probability distribution for all actions ($\pi(a)$) to extract the probability of the picked action ($\pi(\hat{a}_t)$), since $\hat{a}_t$ is a one-hot encoded vector. A product symbol inside a circle is used in Fig. 8 to represent the scalar product.

The complete model is trained by consecutively iterating the value and policy functions with Stochastic Gradient Descent (SGD) to decrease the corresponding loss for each function: a log-loss for the policy and a quadratic loss for the value function, respectively.

The data preparation for this model is similar to DQN. It does not require adapting the data to trajectories of states and actions, as needed for the policy gradient model.

For prediction, in a similar way to the policy gradient model, we use the NN that implements the policy function, simply by selecting the action with the highest probability (without sampling).

Our implementation of the model follows conceptually the schema presented in Fig. 8, however for our final model we apply some minor tricks to reduce the training time. For example, we gather in a single NN the value and policy functions, implementing an NN with the state as input and k different outputs for the value and action, respectively; where k is the number of intrusion labels to predict (one hot encoded). Therefore, we can perform the training of both functions in a single iteration, employing a loss function formed by the sum of the losses of the two original functions.

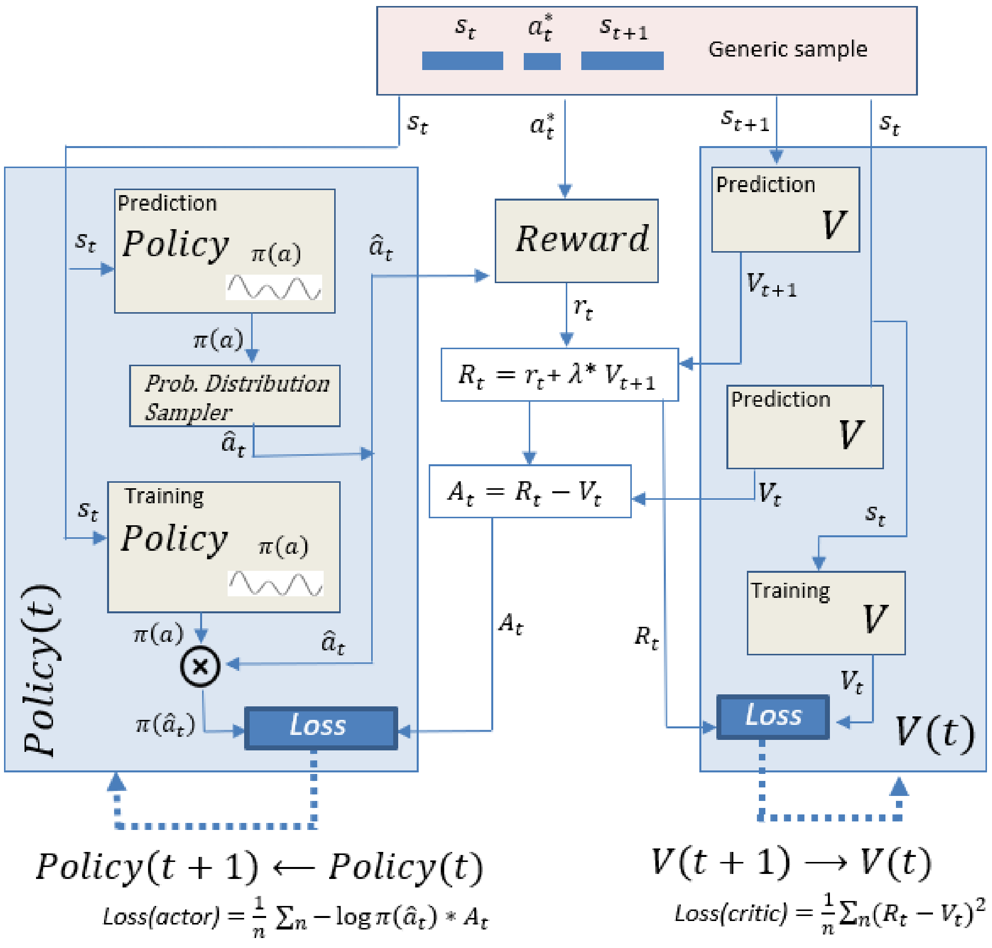


Fig. 8. Schema of the Actor-Critic model during training

4. Results

In this section, we compare the results of applying different machine learning models to the NSL-KDD and AWID datasets. We have selected some of the most common machine learning and deep learning techniques, including Logistic Regression, Support Vector Machine (SVM) with linear kernel and Radial Basis Function (RBF) kernel, K-Nearest Neighbors (KNN), Naive Bayes (NB), Random Forest, Gradient Boosting Machine (GBM), AdaBoost with several weak learners (simple trees and NB), MLP, Convolutional Neural Network (CNN), and our proposed models based in DRL: DQN, DDQN, policy gradient and actor-critic.

All results presented in this paper are based on the full NSL-KDD and AWID test sets (Section 3.1), without any sampling or preparation of an alternative test set taken from the training data. This is an important difference when comparing results with other works, as the unbalanced and difficult prediction structure of the original datasets can be significantly altered by choosing a different test set.

In order to analyze the prediction performance of the different models, we provide the following performance metrics: accuracy, F1 score, precision and recall. We base our definition of these performance metrics on the usually accepted ones (Bhuyan, Bhattacharyya & Kalita, 2014). Due to the unbalanced nature of the datasets (mainly AWID) we will provide more importance to the F1 score which is more suitable in case of imbalance datasets.

To present the results for the two datasets, we first provide the results of the experiments with NSL-KDD dataset followed by the AWID dataset.

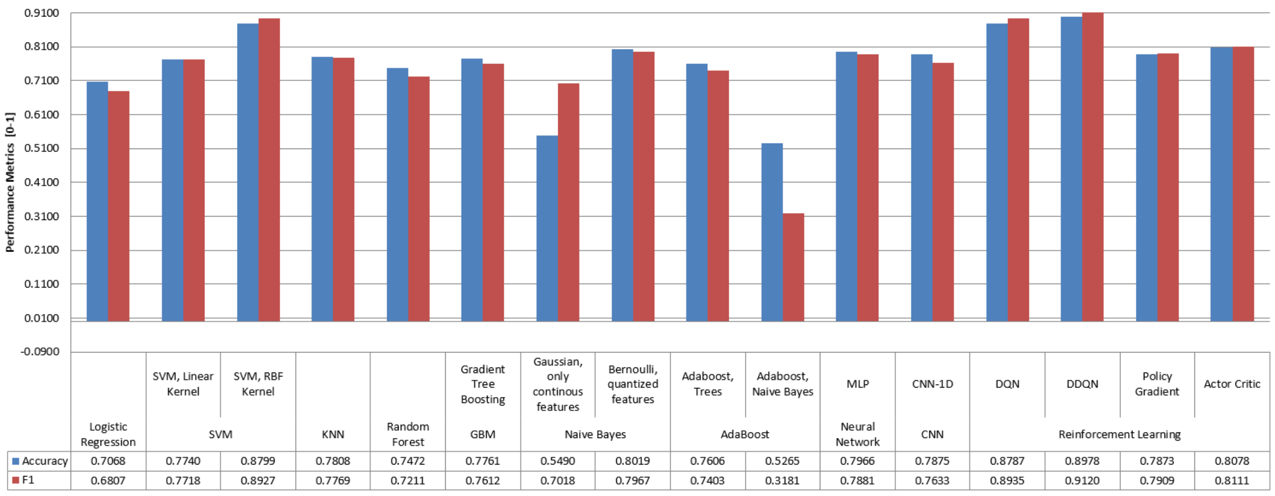
4.1 Results for the NSL-KDD dataset

One of the main contributions of this work is to show the adequacy of the DRL models for intrusion detection in networking. In Fig 9 we present the results for the four DRL models that have been studied (DQN, DDQN, Policy Gradient and Actor Critic) when applied to the NSL-KDD dataset. Fig 9 presents the results in two parts. The upper part presents the raw data in a color-coded way, where the greenest is associated with a better value and the redder with a worse value (comparison of values is applied column-wise). And, the lower part presents only the accuracy and F1 scores (the most significant scores) in a chart; in this case, a variant of Naive Bayes has been eliminated from the graph to make it less cluttered considering the scarce importance of this model in terms of results.

In addition to the detection scores for the DRL models, Fig. 9 shows the performance metrics for all models. We can observe how the DDQN and DQN models produce the best results (considering F1, accuracy and recall), followed by SVM (with RBF kernel) and actor-critic models.

The NSL-KDD dataset presents many challenges for a classifier, with the different composition of the training and test sets being one of the most important. We can also observe in Fig 9 how DDQN stands out in the Recall metric. This metric is very important to guarantee a minimum number of false negatives (samples with an intrusion that are predicted to be normal), which is the main performance metric for an intrusion detection system that tries to identify as many intrusions as possible, considering all of them as critical.

		Test results			
		Accuracy	F1	Precision	Recall
Logistic Regression	Logistic Regression	0.7068	0.6807	0.8955	0.5491
SVM	SVM, Linear Kernel	0.7740	0.7718	0.9073	0.6715
	SVM, RBF Kernel	0.8799	0.8927	0.9081	0.8779
KNN	K-Nearest Neighbors	0.7808	0.7769	0.9233	0.6706
Random Forest	Random Forest	0.7472	0.7211	0.9688	0.5743
GBM	Gradient Tree Boosting	0.7761	0.7612	0.9690	0.6267
Naive Bayes	Gaussian, only continous features	0.5490	0.7018	0.5627	0.9324
	Bernoulli, only discrete features	0.7942	0.7979	0.9046	0.7138
	Bernoulli, all features, quantizing the continous features	0.8019	0.7967	0.9583	0.6818
AdaBoost	Adaboost with Trees	0.7606	0.7403	0.9683	0.5992
	Adaboost with Naive Bayes	0.5265	0.3181	0.8824	0.1940
Neural Network	Neural Net (MLP)	0.7966	0.7881	0.9679	0.6647
CNN	CNN-1D	0.7875	0.7633	0.8094	0.7875
Reinforcement Learning	DQN	0.8787	0.8935	0.8933	0.8937
	DDQN	0.8978	0.9120	0.8944	0.9303
	Policy Gradient	0.7873	0.7909	0.8980	0.7067
	Actor Critic	0.8078	0.8111	0.9203	0.7251

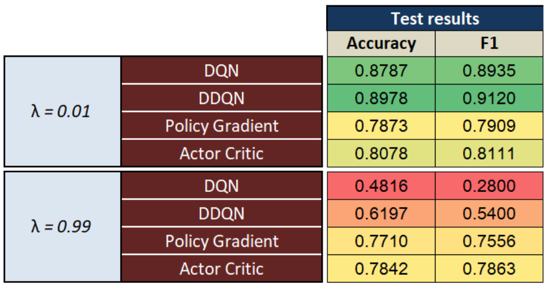


	Logistic Regression	SVM, Linear Kernel (SVM)	SVM, RBF Kernel (SVM)	KNN	Random Forest	Gradient Tree Boosting (GBM)	Gaussian, only continous features (Naive Bayes)	Bernoulli, quantized features (Naive Bayes)	Adaboost, Trees (AdaBoost)	Adaboost, Naive Bayes (AdaBoost)	MLP (Neural Network)	CNN-1D (CNN)	DQN (Reinforcement Learning)	DDQN (Reinforcement Learning)	Policy Gradient (Reinforcement Learning)	Actor Critic (Reinforcement Learning)
🟦 Accuracy	0.7068	0.7740	0.8799	0.7808	0.7472	0.7761	0.5490	0.8019	0.7606	0.5265	0.7966	0.7875	0.8787	0.8978	0.7873	0.8078
🟥 F1	0.6807	0.7718	0.8927	0.7769	0.7211	0.7612	0.7018	0.7967	0.7403	0.3181	0.7881	0.7633	0.8935	0.9120	0.7909	0.8111

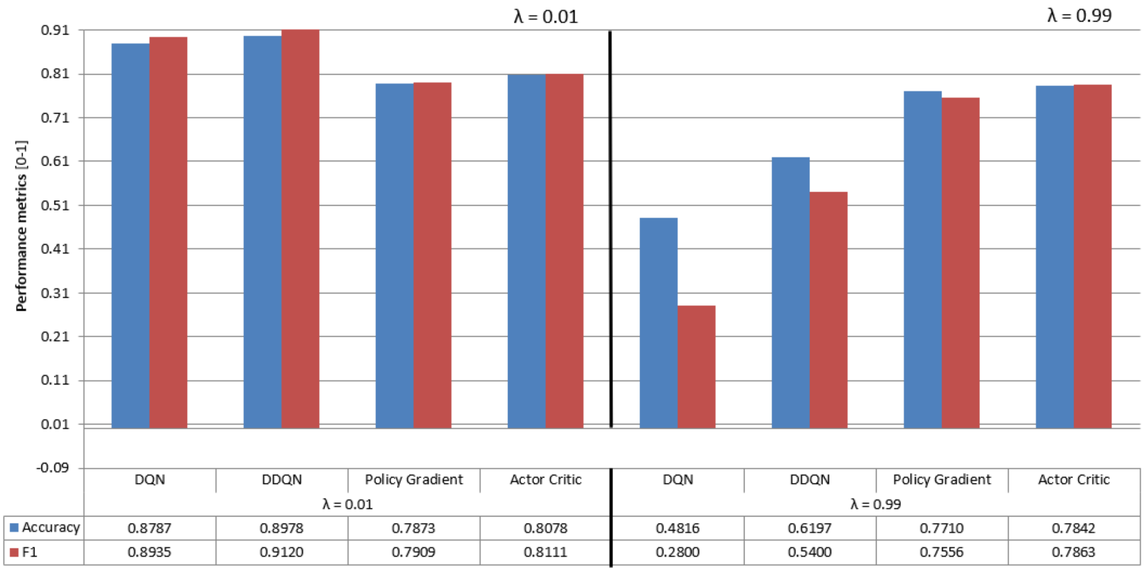
Fig. 9. Performance scores for all models (NSL-KDD dataset)

As noted in Section 3.2, the discount factor (λ) considered for the DRL algorithms can have a significant influence on the results. To study this influence, Fig. 10 shows the impact of different values for the discount factor used in the DRL models. The impact is critical for DQN and DDQN and less important for policy gradient and actor-critic models. We obtain better results for very low values of the discount factor, as explained in Section 3.2.2.1. This is clearer for the DQN and DDQN models than for the policy gradient model, since the latter optimizes the network parameters based on a sequence (episode) of states/actions, implicitly performing an optimization on an averaged sum of rewards, whereas the DQN and DDQN models are based on single state and action pairs.

Fig 10 also presents the data in two sections, with the upper section presenting the raw data with a color-code similar to that of Fig 9, and the lower section showing the same data in a chart format. This same data presentation structure is used for Fig 11 and 12.



		Test results	
		Accuracy	F1
λ = 0.01	DQN	0.8787	0.8935
	DDQN	0.8978	0.9120
	Policy Gradient	0.7873	0.7909
	Actor Critic	0.8078	0.8111
λ = 0.99	DQN	0.4816	0.2800
	DDQN	0.6197	0.5400
	Policy Gradient	0.7710	0.7556
	Actor Critic	0.7842	0.7863

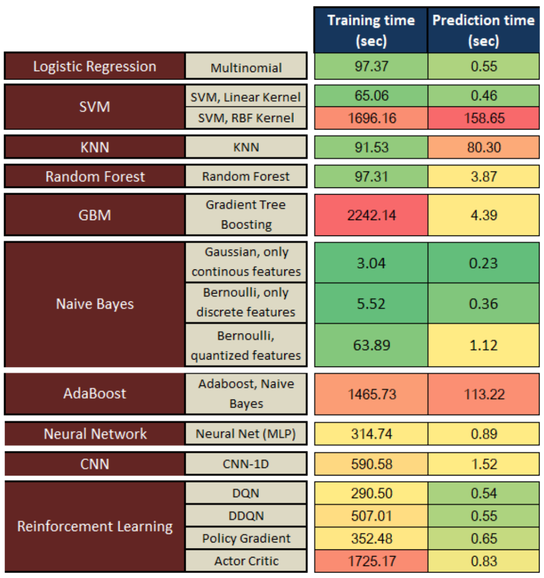


	λ = 0.01				λ = 0.99			
	DQN	DDQN	Policy Gradient	Actor Critic	DQN	DDQN	Policy Gradient	Actor Critic
🟦 Accuracy	0.8787	0.8978	0.7873	0.8078	0.4816	0.6197	0.7710	0.7842
🟥 F1	0.8935	0.9120	0.7909	0.8111	0.2800	0.5400	0.7556	0.7863

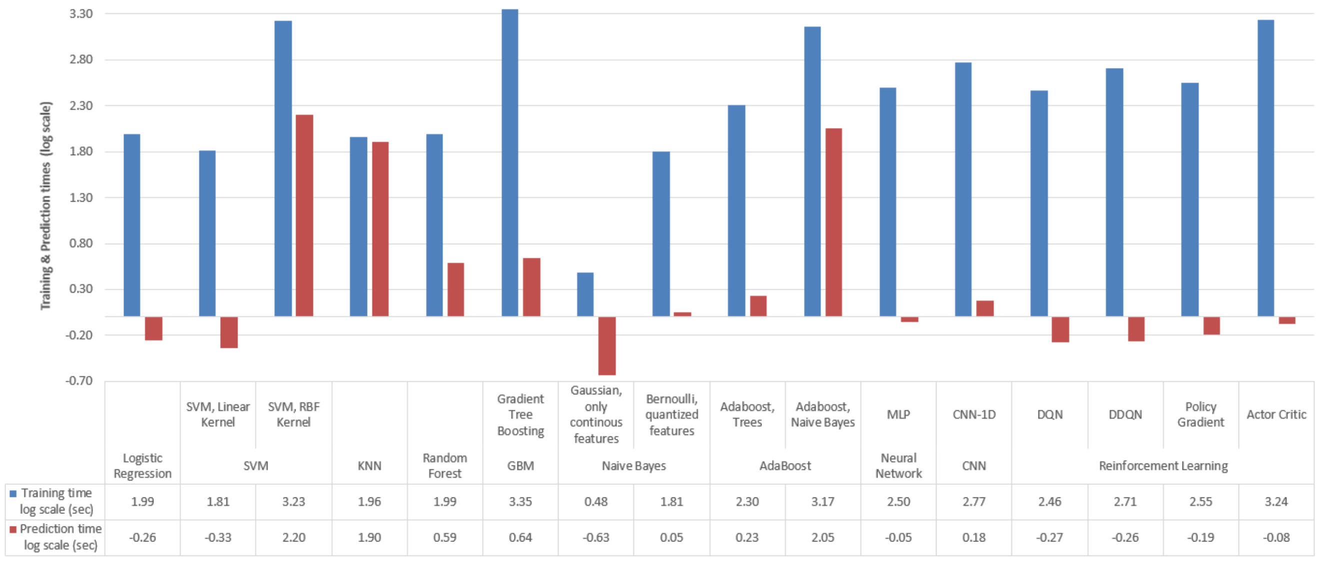
Fig. 10. Comparison of performance scores for different discount factors (NSL-KDD dataset)

As presented earlier, one of the advantages of the DRL models is that once trained, the resulting policy function, which provides the correct intrusion label (action) for a specific state (intrusion features), is actually a simple neural network which can be extremely fast for the inference (prediction) stage and, therefore, suitable to be used in an industrial production environment. This behavior can be checked in Fig. 11, where training and prediction times for all models are shown. We can appreciate how the prediction times for all DRL models are really very small compared to the second model with better results (SVM-RBF). For training times, the values are also smaller for the DRL models, with the exception of the actor-critic model, which requires a considerable amount of training time. As expected, the best training times are obtained for the linear models (logistic regression and linear SVM) together with Random Forest and Naive Bayes.

In Fig 11, the color-coded raw data table (upper part) is complemented with an associated chart (lower part) of the training and prediction times in logarithmic scale. Considering the large range of values, the logarithmic scale allows a more suitable view of the data. In the chart of Fig 11, the smallest values (positive or negative) correspond to a better value in terms of training or prediction times.



		Training time (sec)	Prediction time (sec)
Logistic Regression	Multinomial	97.37	0.55
SVM	SVM, Linear Kernel	65.06	0.46
	SVM, RBF Kernel	1696.16	158.65
KNN	KNN	91.53	80.30
Random Forest	Random Forest	97.31	3.87
GBM	Gradient Tree Boosting	2242.14	4.39
Naive Bayes	Gaussian, only continous features	3.04	0.23
	Bernoulli, only discrete features	5.52	0.36
	Bernoulli, quantized features	63.89	1.12
AdaBoost	Adaboost, Naive Bayes	1465.73	113.22
Neural Network	Neural Net (MLP)	314.74	0.89
CNN	CNN-1D	590.58	1.52
Reinforcement Learning	DQN	290.50	0.54
	DDQN	507.01	0.55
	Policy Gradient	352.48	0.65
	Actor Critic	1725.17	0.83



	Logistic Regression	SVM		KNN	Random Forest	GBM	Naive Bayes		AdaBoost		Neural Network	CNN	Reinforcement Learning			
		SVM, Linear Kernel	SVM, RBF Kernel			Gradient Tree Boosting	Gaussian, only continous features	Bernoulli, quantized features	Adaboost, Trees	Adaboost, Naive Bayes	MLP	CNN-1D	DQN	DDQN	Policy Gradient	Actor Critic
■ Training time log scale (sec)	1.99	1.81	3.23	1.96	1.99	3.35	0.48	1.81	2.30	3.17	2.50	2.77	2.46	2.71	2.55	3.24
■ Prediction time log scale (sec)	-0.26	-0.33	2.20	1.90	0.59	0.64	-0.63	0.05	0.23	2.05	-0.05	0.18	-0.27	-0.26	-0.19	-0.08

Fig. 11. Training and prediction times for all models (NSL-KDD dataset)

It is important to mention here several details about the different ML models applied in the study. For SVM with linear kernel, we have used the primal solution which provides a much faster implementation in our specific case (high number of samples and small number of features). We have implemented the MLP with three hidden layers with 1024, 512 and 128 nodes. Naive Bayes (NB) was applied with different features arrangements: a Gaussian NB using only the continuous features, a Bernoulli NB using only the discrete features and finally a Bernoulli NB using all features. In order to apply the Bernoulli NB to the continuous features, we needed to transform them from continuous to discrete intervals (quantization). Considering the CNN model (Goodfellow, Bengio & Courville, 2016), we have applied the one-dimensional CNN due to the nature of the dataset.

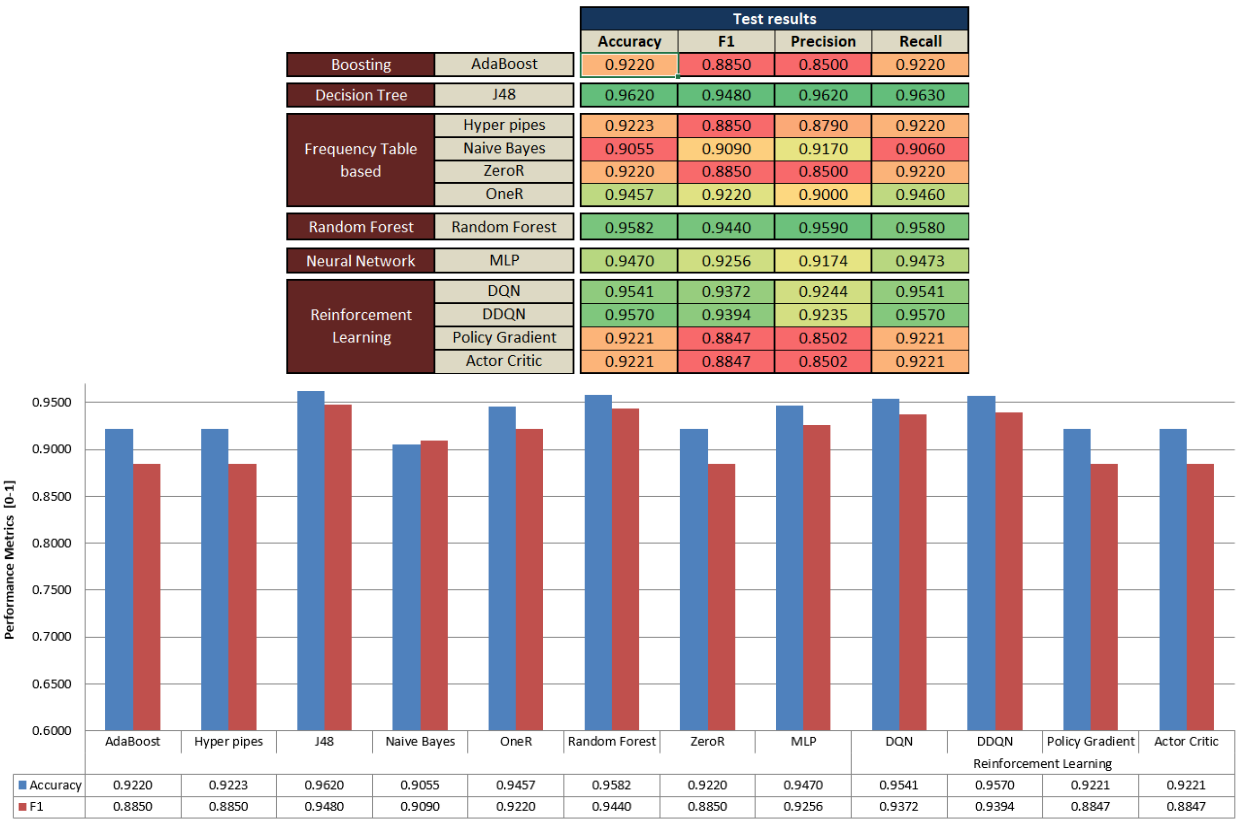
4.2 Results for the AWID dataset

AWID is an extremely unbalanced dataset, useful to test the performance of an intrusion detector (Section 3.1.2). Fig 12 presents the performance results of the different DRL models studied in this work together with a wide range of alternative ML models. The results for the alternative ML models have been extracted from the original work of the research team that created the data set (Kolias et al., 2016), where in addition to an analysis of the reasons for its creation and a detailed description of its composition, it is presented a complete study of the expected classification results using different ML models.

For the AWID dataset we have a multi-class classification problem. In a multi-class classification, the results can be provided in two possible ways: aggregated and one vs. rest.

In the case of one vs. rest, we consider each particular class (label) against all other classes, reducing it to a sequence of binary classifications (one for each particular class). In the case of aggregated results, we provide a single result which is a summary (average) for all classes. There are different possibilities for the aggregation (micro, macro, samples, weighted), with different averaging methods.

The performance metrics provided in Fig 12 are aggregated metrics employing a weighted average for the F1, precision and recall as shown in (Pedregosa et al., 2011). We can observe how the DDQN model presents excellent results for the Accuracy, F1 and Recall metrics. For this dataset, the Random Forest and Decision Tree (J48) models obtain the best results. It is important to mention that DDQN excels again in the Recall metric for this dataset, which, as mentioned for the results of NSL-KDD (Section 4.1), is a critical metric for an intrusion detection algorithm that attempts to reduce false negatives (intrusions that are not detected).

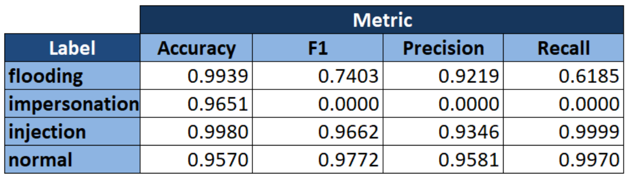


		Test results			
		Accuracy	F1	Precision	Recall
Boosting	AdaBoost	0.9220	0.8850	0.8500	0.9220
Decision Tree	J48	0.9620	0.9480	0.9620	0.9630
Frequency Table based	Hyper pipes	0.9223	0.8850	0.8790	0.9220
	Naive Bayes	0.9055	0.9090	0.9170	0.9060
	ZeroR	0.9220	0.8850	0.8500	0.9220
	OneR	0.9457	0.9220	0.9000	0.9460
Random Forest	Random Forest	0.9582	0.9440	0.9590	0.9580
Neural Network	MLP	0.9470	0.9256	0.9174	0.9473
Reinforcement Learning	DQN	0.9541	0.9372	0.9244	0.9541
	DDQN	0.9570	0.9394	0.9235	0.9570
	Policy Gradient	0.9221	0.8847	0.8502	0.9221
	Actor Critic	0.9221	0.8847	0.8502	0.9221

	AdaBoost	Hyper pipes	J48	Naive Bayes	OneR	Random Forest	ZeroR	MLP	DQN	DDQN	Policy Gradient	Actor Critic
🟦 Accuracy	0.9220	0.9223	0.9620	0.9055	0.9457	0.9582	0.9220	0.9470	0.9541	0.9570	0.9221	0.9221
🟥 F1	0.8850	0.8850	0.9480	0.9090	0.9220	0.9440	0.8850	0.9256	0.9372	0.9394	0.8847	0.8847

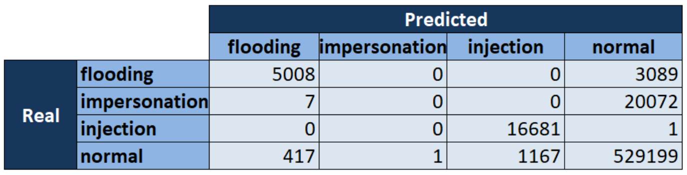
Fig. 12. Performance scores for all models (AWID dataset)

Considering the one vs. rest metrics for the 4 class values of the AWID dataset, the results are provided in Table I. In this table, each row corresponds to the scores obtained when considering a binary classification between a label (the one associated with the row) and the rest of the labels. Even considering the poor results for one of the class values: impersonation, the rest presents very good results. It is important to keep in mind the strong unbalanced nature of the dataset, and how easy is for a classifier, under these conditions, to adopt the majority class as the unique classification result. This only occurs for one of the class values (impersonation), as can be seen in detail in Table II that shows the confusion matrix (Bhuyan, Bhattacharyya & Kalita, 2014) for the DDQN model applied to the AWID dataset with 4 class values. The number of true positives in Table II is high for all class values except for the impersonation attack, with no true positives (that is the reason for the zero value of F1, precision and recall, in Table I) and, where almost all predictions have fallen within the normal value.



	Metric			
Label	Accuracy	F1	Precision	Recall
flooding	0.9939	0.7403	0.9219	0.6185
impersonation	0.9651	0.0000	0.0000	0.0000
injection	0.9980	0.9662	0.9346	0.9999
normal	0.9570	0.9772	0.9581	0.9970

Table I. One vs. rest performance metrics for the DDQN model (AWID dataset)



		Predicted			
		flooding	impersonation	injection	normal
Real	flooding	5008	0	0	3089
	impersonation	7	0	0	20072
	injection	0	0	16681	1
	normal	417	1	1167	529199

Table II. Confusion matrix for the DDQN model (AWID dataset)

4.3 Summary of results

The DDQN model (which is the DRL model that presents best results) has a comparable detection performance (accuracy, F1, precision and recall), and, in some cases, better than alternative SOTA ML models (e.g. Random Forest, Decision Trees, SVM, Naïve Bayes…).

The results were obtained with two different IDS datasets, and in both cases our model based on DDQN has been in the set of the best solution models, while the alternative SOTA model, for each dataset, has been very different in each case (SVM and Decision Trees for NSL-KDD and AWID respectively). The conclusion is that our model provides a more robust solution in different scenarios.

It is particularly important to mention the excellent results of DDQN in the recall metric, which is crucial for an intrusion detection algorithm, since we want to reduce at a minimum the number of false negatives.

In addition to demonstrating that DRL models can be applied to intrusion detection problems with a labeled dataset, the classifiers obtained (once they are trained) are a simple and fast neural network that can be deployed in modern high-performance and distributed computing environments (e.g. Tensorflow) (Abadi et al., 2016). In particular, the DDQN model, at test time, provides much smaller prediction times than the best SOTA models studied for this work.

Considering the results presented in Fig. 10, we can conclude that the application of DRL models to a scenario with a labelled dataset depends to a large extent on the choice of the value of the discount factor. This is an unexpected and significant discovery, since it seems that the fact of not interacting with a live environment (which means that the feedback loop caused by the impact of the actions on the environment is broken), we must be more conservative in each update of our policy function, making the convergence slower but more stable. This effect is particularly important for the DQN and DDQN models due to the reasons provided in Section 4.1.

We have implemented all the models in python using the scikit-learn package (Pedregosa et al., 2011), except all linear models (including linear-SVM and Logistic Regression), MLP, CNN and all DRL models for which we have used Tensorflow. All computations have been performed in a PC with an Intel i7 CPU and 16GB RAM.

5. Conclusion

As a summary, the contributions of the paper are: (1) New algorithm that improves the results of intrusion detection compared to the existing techniques of machine learning and deep learning. (2) Intrusion detection algorithm based on an extremely simple and fast policy network, especially suitable for demanding applications in modern data networks that require a rapid response. (3) The resulting model is suitable for on-line learning, which is necessary for data networks with changing environments. (4) Novel application of DRL for supervised learning. (5) The optimization process is driven by a rewards function that is not required to be differentiable, which makes it more flexible an applicable to all kind of problems.

We provide a comparison study of four DRL algorithms (DQN, DDQN, Policy gradient and actor-critic) and how they can be applied to a dataset labeled with intrusions instead of interacting with a live network environment. An additional analysis is provided comparing these algorithms with several alternative machine learning models, considering three performance aspects: (1) prediction scores, (2) training and (3) prediction times, and using two different intrusion detection datasets (NSL-KDD and AWID) to facilitate the generalization of the results.

The best DRL algorithm (DDQN) has a detection performance (measured by several performance metrics: accuracy, F1, precision and recall) comparable, and, in some cases, better than a full range of SOTA ML models (e.g. Random Forest, Decision Trees, SVM, Naive Bayes…). In addition, DDQN, and in general DRL methods, present and important advantage in terms of significantly reduced prediction times, which makes them very suitable for on-line detection and new highly demanding network services (e.g. IoT networks).

In addition to the novel application of a DRL framework to a dataset of logged features and associated class labels (instead of a live environment capable of responding in real time to the actions of the algorithm), another important contribution of this work is to show the importance of the discount factor parameter that regulates the speed of convergence of the algorithm, being especially important to have a small value for this parameter for the convergence of the DQN and DDQN algorithms, under the restrictions imposed by this work. Another contribution of this work is to show the necessary data preparation that is required to apply the DRL models to a labeled dataset, and to propose a way of doing this preparation considering the specificities of the different models. Finally, our research on reward functions (distance between the predicted and ground-truth labels) has produced the surprising conclusion that the simple 1/0 reward function produces better results than more sophisticated alternatives, e.g. cross-entropy and categorical hinge (section 3.2.2.1).

As future work, we plan to investigate the application of new DRL algorithms, specifically the application of multi-agent and adversarial models for DRL which can be applicable to intrusion detection problems (Pinto et al. 2017; Busoniu, Babuska & De Schutter, 2010).

References

Abadi M. et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467v2 [cs.DC]

Abdulhammed R. et al. (2018). Effective Features Selection and Machine Learning Classifiers for Improved Wireless Intrusion Detection. 2018 International Symposium on Networks, Computers and Communications (ISNCC), Rome, pp. 1-6. https://doi.org/10.1109/ISNCC.2018.8530969

Akazaki, T. et al. (2018). Falsification of cyber-physical systems using deep reinforcement learning. In International Symposium on Formal Methods (pp. 456-465). Springer, Cham. https://doi.org/10.1007/978-3-319-95582-7_27

Arulkumaran K. et al. (2017). Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26-38. https://doi.org/10.1109/MSP.2017.2743240

Bartlett P. L. & Baxter J. (2011). Infinite-Horizon Policy-Gradient Estimation. arXiv:1106.0665 [cs.AI]

Berman D.S. et al. (2019). A survey of deep learning methods for cyber security. Information, 10(4), 122. https://doi.org/10.3390/info10040122

Bhattacharjee P.S., Fujail A.K.Md & Begum S.A. (2017). A Comparison of Intrusion Detection by K-Means and Fuzzy C-Means Clustering Algorithm Over the NSL-KDD Dataset, *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC),* Coimbatore, pp. 1-6. https://doi.org/10.1109/ICCIC.2017.8524401

Bhuyan, M.H.; Bhattacharyya, D.K. & Kalita, J.K. (2014). Network Anomaly Detection: Methods, Systems and Tools. IEEE Communications Surveys & Tutorials; IEEE: Piscataway, NJ, USA, Volume 16, pp. 303–336. https://doi.org/10.1109/SURV.2013.052213.00046

Boutaba R. et al. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. Journal of Internet Services and Applications (2018) 9:16. https://doi.org/10.1186/s13174-018-0087-2

Busoniu L., Babuska R., & De Schutter B. (2010). Multi-agent Reinforcement Learning: An Overview. In: Srinivasan D., Jain L.C. (eds) Innovations in Multi-Agent Systems and Applications - 1. Studies in Computational Intelligence, vol 310. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-14435-6_7

Chalapathy R. & Chawla S. (2019). Deep Learning for Anomaly Detection: A Survey. arXiv:1901.03407 [cs.LG]

Chen Z. et al. (2018). Autoencoder-based network anomaly detection. *2018 Wireless Telecommunications Symposium (WTS)*, Phoenix, AZ, pp. 1-5. https://doi.org/10.1109/WTS.2018.8363930

da Costa K.A.P. et al. (2019). Internet of Things: A survey on machine learning-based intrusion detection approaches. Computer Networks. Vol. 151. pp. 147-157. https://doi.org/10.1016/j.comnet.2019.01.023

Deokar, B., & Hazarnis, A. (2012). Intrusion detection system using log files and reinforcement learning. International Journal of Computer Applications, 45(19), 28-35.

Dhanabal L. & Shantharajah S.P. (2015). 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, Issue 6

Feng, M., & Xu, H. (2017). Deep reinforcement learning based optimal defense for cyber-physical system in presence of unknown cyber-attack. *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Honolulu, HI, pp. 1-8. https://doi.org/10.1109/SSCI.2017.8285298

Goodfellow I., Shlens J. & Szegedy Ch. (2015). Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

Goodfellow I., Bengio Y. & Courville A. (2016). Deep Learning. Book. MIT Press, Ch 9. http://www.deeplearningbook.org/

Grondman I. et al. (2012). A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1291-1307. https://doi.org/10.1109/TSMCC.2012.2218595

Hussain F. et al. (2019). Machine Learning in IoT Security: Current Solutions and Future Challenges. arXiv:1904.05735v1 [cs.CR]

Ibrahim L.M. et al, (2013). A comparison study for intrusion database (KDD99, NSL-KDD) based on self-organization map (SOM) artificial neural network. Journal of Engineering Science and Technology, School of Engineering, Taylor’s University, Vol. 8, No. 1, pp. 107 – 119. https://doaj.org/article/fa185b23f07c4a54b11d9997fd95958a

Ingre B, & Yadav A. (2015). Performance Analysis of NSL-KDD dataset using ANN. 2015 International Conference on Signal Processing and Communication Engineering Systems, Guntur, 2015, pp. 92-96. https://doi.org/10.1109/SPACES.2015.7058223

Javaid A. et al. (2016). A Deep Learning Approach for Network Intrusion Detection System. Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)(BICT'15), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Belgium. http://dx.doi.org/10.4108/eai.3-12-2015.2262516

Kamel S.O.M et al. (2016). AdaBoost Ensemble Learning Technique for Optimal Feature Subset Selection. International Journal of Computer Networks and Communications Security vol. 4, no. 1, pp. 1–11

Kim K. & Aminanto M.E. (2017). Deep learning in intrusion detection perspective: Overview and further challenges. 2017 International Workshop on Big Data and Information Security (IWBIS), Jakarta, pp. 5-10. https://doi.org/10.1109/IWBIS.2017.8275095

Kolias C., et al (2016). Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset. IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 184-208. https://doi.org/10.1109/COMST.2015.2402161

Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., & Lloret, J. (2017). Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in IoT. Sensors, 17 (9), 1967. 2017 https://doi.org/10.3390/s17091967

Malialis, K. (2014). Distributed Reinforcement Learning for Network Intrusion Response. PhD Thesis, University of York. UK. http://etheses.whiterose.ac.uk/id/eprint/8109

Moshkov N. (2017). Program for Classification of Intrusion Type on Local Computer System. Master Thesis. National Research University Higher School of Economics. Moscow. Russia. https://www.hse.ru/en/edu/vkr/206739508

Mnih V. et al. (2013). Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG]

Nguyen T.T. & Reddi V.J. (2019). Deep Reinforcement Learning for Cyber Security, arXiv:1906.05799 [cs.CR]

Nisioti A. et al. (2018). From Intrusion Detection to Attacker Attribution: A Comprehensive Survey of Unsupervised Methods. *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3369-3388. https://doi.org/10.1109/COMST.2018.2854724

Nivaashini.M. & Thangaraj.P. (2019) State-Of-The-Art Machine Learning and Deep Learning: Evolution of Intelligent Intrusion Detection System against Wireless Network (WiFi) Attacks in Internet of Things (Iot). International Journal of Innovative Technology and Exploring Engineering (IJITEE). Vol-8 Issue-3.

Panda M. Abraham A. & Patra M. R. (2010). Discriminative Multinomial Naïve Bayes for Network Intrusion Detection. *2010 Sixth International Conference on Information Assurance and Security*, Atlanta, GA, 2010, pp. 5-10. https://doi.org/10.1109/ISIAS.2010.5604193

Patil D.R. & Pattewar T.M. (2014). A Comparative Performance Evaluation of Machine Learning-Based NIDS on Benchmark Datasets. International Journal of Research in Advent Technology, Vol.2, No.2, April 2014 E-ISSN: 2321-9637.

Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12 (2011), pp. 2825-2830. http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

Pinto L., et al. (2017). Robust Adversarial Reinforcement Learning. arXiv:1703.02702v1 [cs.LG]

Qin Y. et al. (2018). Attack Detection for Wireless Enterprise Network: a Machine Learning Approach. *IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, Qingdao, pp. 1-6. https://doi.org/10.1109/ICSPCC.2018.8567797

Rezvy S. et al. (2019). An efficient deep learning model for intrusion classification and prediction in 5G and IoT networks. 2019 53rd Annual Conference on Information Sciences and Systems (CISS), Baltimore, USA, pp. 1-6. https://doi.org/10.1109/CISS.2019.8693059

Ring M. et al. (2019). A survey of network-based intrusion detection data sets. Computers & Security. Vol. 86. pp. 147-167. https://doi.org/10.1016/j.cose.2019.06.005

Servin A. (2007). Towards Traffic Anomaly Detection via Reinforcement Learning and Data Flow. Department of Computer Science, University of York. UK. https://www.cs.york.ac.uk/yds/pub/07/proceedings_07/proceedings_07/11/author.11.pdf

Servin A. (2009). Multi-Agent Reinforcement Learning for Intrusion Detection. PhD Thesis, University of York. UK. http://etheses.whiterose.ac.uk/id/eprint/690

Shone N. et al. (2018). A Deep Learning Approach to Network Intrusion Detection. IEEE Transactions on Emerging Topics in Computational Intelligence, Vol. 2, No. 1, pp. 41-50. https://doi.org /10.1109/TETCI.2017.2772792

Sukhanov A.V., Kovalev S.M. & Stýskala V. (2015). Advanced Temporal-Difference Learning for Intrusion Detection. IFAC-PapersOnLine, Volume 48, Issue 4, 2015, Pages 43-48. https://doi.org/10.1016/j.ifacol.2015.07.005

Sutton R.S. (1988). Learning to Predict by the Methods of Temporal Differences. Machine Learning. Vol 3, Issue 1, pp. 9–44. https://doi.org/10.1007/BF00115009

Sutton R. S. & Barto A. G. (1998). Reinforcement Learning: An Introduction. A Bradford Book; Second Edition. https://mitpress.mit.edu/books/reinforcement-learning-second-edition

Tavallaee M. et al. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, ON, 2009, pp. 1-6. https://doi.org/10.1109/CISDA.2009.5356528

Thanthrige U.S.K.P.M., Samarabandu J.& Wang X. (2016). Machine learning techniques for intrusion detection on public dataset. IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Vancouver, BC, pp. 1-4. https://doi.org/10.1109/CCECE.2016.7726677

Thomas R. & Pavithran D. (2018). A Survey of Intrusion Detection Models based on NSL-KDD Data Set. ," *2018 Fifth HCT Information Technology Trends (ITT)*, Dubai, UAE, pp. 286-291. https://doi.org/10.1109/CTIT.2018.8649498

Tsai Ch-F. et al. (2009) Intrusion detection by machine learning: A review. Expert Systems with Applications. Vol 36. Issue 10. pp. 11994-12000. https://doi.org/10.1016/j.eswa.2009.05.029

Van Hasselt H. et al. (2015). Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs.LG].

Wang M. et al. (2018). Machine Learning for Networking: Workflow, Advances and Opportunities. IEEE Network, vol. 32, no. 2, pp. 92-99. https://doi.org/10.1109/MNET.2017.1700200

Wang S. et al. (2019). Intrusion Detection for WiFi Network: A Deep Learning Approach. International Wireless Internet Conference. WICON 2018: Wireless Internet. pp. 95-104, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 264. Springer, Cham. https://doi.org/10.1007/978-3-030-06158-6_10

Wiering M.A. et al. (2011). Reinforcement learning algorithms for solving classification problems. *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Paris, 2011, pp. 91-96. https://doi.org/10.1109/ADPRL.2011.5967372

Woo J., Song J. & Choi Y. (2019). Performance Enhancement of Deep Neural Network Using Feature Selection and Preprocessing for Intrusion Detection. *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Okinawa, Japan, pp. 415-417. https://doi.org/10.1109/ICAIIC.2019.8668995

Xu X. & Xie T. (2005). A Reinforcement Learning Approach for Host-Based Intrusion Detection Using Sequences of System Calls. In: Huang DS., Zhang XP., Huang GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, vol 3644. Springer, Berlin. https://doi.org/10.1007/11538059_103

Xu X. (2006) A Sparse Kernel-Based Least-Squares Temporal Difference Algorithm for Reinforcement Learning. Advances in Natural Computation. ICNC 2006. Lecture Notes in Computer Science, vol 4221. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11881070_8

Xu, X., & Luo, Y. (2007). A kernel-based reinforcement learning approach to dynamic behavior modeling of intrusion detection. International Symposium on Neural Networks (pp. 455-464). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-72383-7_54

Xu X. (2010). "Sequential anomaly detection based on temporal-difference learning: Principles, models and case studies", Applied Soft Computing, Volume 10, Issue 3, 2010, Pages 859-867. https://doi.org/10.1016/j.asoc.2009.10.003

Yavanoglu O. & Aydos M. (2017). A review on cyber security datasets for machine learning algorithms. *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, pp. 2186-2193. https://doi.org/10.1109/BigData.2017.8258167

Zarpelo, B.B. et al. (2017). A survey of intrusion detection in Internet of Things. Journal of Network and Computer Applications. Vol. 84. 2017. pp. 25-37. https://doi.org/10.1016/j.jnca.2017.02.009