IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 14, 2021, accepted November 5, 2021, date of publication November 11, 2021,
date of current version November 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3127689

Network Intrusion Detection Based on Extended RBF Neural Network With Offline Reinforcement Learning

MANUEL LOPEZ-MARTIN[1], (Senior Member, IEEE), ANTONIO SANCHEZ-ESGUEVILLAS[1], (Senior Member, IEEE), JUAN IGNACIO ARRIBAS[1,2], AND BELEN CARRO[1]

[1]Department of TSyCeIT, ETSIT, University of Valladolid, 47011 Valladolid, Spain
[2]Castilla-Leon Neuroscience Institute, University of Salamanca, 37007 Salamanca, Spain

Corresponding author: Manuel Lopez-Martin (manuel.lopezm@uva.es)

This work was supported in part by the Proyectos de I+D+i ≪Retos investigación≫, Programa Estatal de I+D+i Orientada a los Retos de la Sociedad, Plan Estatal de Investigación Científica, Técnica y de Innovación 2017–2020 under Grant RTI2018-098958-B-I00; in part by the Spanish Ministry for Science, Innovation and Universities; in part by the Agencia Estatal de Investigación (AEI); and in part by the Fondo Europeo de Desarrollo Regional (FEDER).

ABSTRACT Network intrusion detection focuses on classifying network traffic as either normal or attack carrier. The classification is based on information extracted from the network flow packets. This is a complex classification problem with unbalanced datasets and noisy data. This work extends the classic radial basis function (RBF) neural network by including it as a policy network in an offline reinforcement learning algorithm. With this approach, all parameters of the radial basis functions (along with the network weights) are learned end-to-end by gradient descent without external optimization. We further explore how additional dense hidden-layers, and the number of radial basis kernels influence the results. This novel approach is applied to five prominent intrusion detection datasets (NSL-KDD, UNSW-NB15, AWID, CICIDS2017 and CICDDOS2019) achieving better performance metrics than alternative state-of-the-art models. Each dataset provides different restrictions and challenges allowing a better validation of results. Analysis of the results shows that the proposed architectures are excellent candidates for designing classifiers with the constraints imposed by network intrusion detection. We discuss the importance of dataset imbalance and how the proposed methods may be critically important for unbalanced datasets.

INDEX TERMS Communication system security, intrusion detection, neural networks, radial basis function networks.

I. INTRODUCTION

Network intrusion detection (NID) focuses on classifying network traffic as either normal or attack carrier. The classification is based on information extracted from the network flow packets. It is a major problem for modern data networks with an active research community currently focused on applying machine learning and deep learning models [1]–[4]. Radial Basis Function Neural Networks (RBFNN) [5] are variants of the feed-forward Neural Network (NN) architecture. RBFNNs has been used for NID in their basic configuration. A basic RBFNN classifier is made of a single hidden-layer NN where each neuron implements a radial basis activation function (RBF), usually a Gaussian function. In addition to the network weights, these networks have two additional types of parameters that need to be optimized: (a) the center of the RFB functions, and (b) its dispersion/scaling parameter. The optimization of these three types of parameters: centers, dispersion factors and weights, are generally carried out following specific methods for each of them [6], causing a complex training process. The optimization methods used for estimating centers and dispersion factors are usually clustering techniques (e.g., k-means) or generic optimization methods, such as: Immune Radial Basis Function (IRBF), Particle swarm optimization (PSO) and Quantum PSO (QPSO). For network weights, the optimization methods used are least squares or gradient descent. This nonintegrated and laborious training process makes it difficult to increase the complexity

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Esposito.

of the network, resulting in the use of simple network architectures consisting of a single hidden layer network (basic RBFNN).

We propose a novel extension of the RBFNN basic model. The RBFNN basic architecture is based on a single hidden-layer where each neuron implements a radial basis activation function e.g., Gaussian. These functions are also called radial kernels, and are defined by two parameters: a center and a scaling/dispersion parameter [6]. These parameters are usually optimized (learned from data) separately and with ad-hoc techniques, such as: clustering techniques (e.g., k-means) or generic optimization methods (e.g., PSO, Quantum PSO). The weights between the RBF layer and the output layer are also separately optimized using least squares or gradient descent. Therefore, the learning process for the basic RBFNN consists of optimizing three types of parameters (centers, dispersion/scaling, and weights) within three distinct and separate learning tasks [6].

In this work we address the above mentioned problems of RBFNN by extending its basic architecture as follows:

(a) **End-to-end learning** of the entire set of parameters using gradient descent; reducing three separate optimization processes, one for each of the model parameter types (weights, centers, and dispersion parameters), to a single optimization mechanism driven by minimization of the classification error (loss function).

(b) An **alternative optimization scheme** for the network parameters based on **offline Deep Reinforcement Learning (DRL)** principles by including the entire RBFNN network as the policy network of a reinforcement learning model; in this way, the loss function is replaced by a more generic reward function [7].

(c) **Increase the complexity of the RBFNN network** by including additional network layers with different configurations, which is made possible by the end-to-end training of the entire network.

This work shows that the basic RBFNN, that has already obtained good classification results for NID, can be successfully extended to more complex architectures while maintaining its basic properties of clustering the feature space into separate groups with an indicator of the level of membership in that group. The group membership score is produced by the RBF function which provides a value that depends on the distance between the inputs and each cluster center. It is a useful representation learning technique and is even better when the parameters that define the location of the groups (centers and dispersion factors) are learned with the ultimate goal of minimizing the classification loss function, which is achievable by the simultaneous end-to-end training of all model parameters.

This work explores the extension of the RBFNN model considering two separate learning frameworks:

(a) A **classic supervised learning framework**, where the complete set of model parameters are optimized by minimizing a loss function (e.g., cross-entropy, hinge-loss) related with the classification error.

TABLE 1. List of abbreviations.

Abbreviation	Full Form
AUC	Area Under the Curve
AWID	Aegean Wi-Fi Intrusion Dataset
CE	Cross Entropy
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
DL	Deep Learning
DQN	Deep Q-learning Network
DDQN	Double DQN
DRL	Deep Reinforcement Learning
ELM	Extreme Learning Machine
FC	Fully Connected
FN	False Negative
FP	False Positive
GBM	Gradient Boosting Machine
HS	Hinge Loss
ID	Intrusion Detection
IRBF	Immune Radial Basis Function
KA	Kernel Approximation
KDD	Knowledge Discovery and Data Mining
LI	Linear
LM	Linear Model
LR	Logistic Regression
ML	Machine Learning
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology
NID	Network Intrusion Detection
NN	Neural Network
PCA	Principal Component Analysis
PSO	Particle Swarm Optimization
QPSO	Quantum PSO
RBF	Radial Basis Function
RBFNN	Radial Basis Function Neural Network
ReLU	Rectified Linear Unit
RF	Random Forest
RL	Reinforcement learning
ROC	Receiver Operating Characteristics
SGD	Stochastic Gradient Descent
SM	SoftMax
SOM	Self-Organizing Map
SOTA	State Of The Art
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
UNSW	University of New South Wales
XAI	eXplainable Artificial Intelligence

(b) A **reinforcement learning framework**, with two elements: an agent (implementing a policy) and an environment. The policy produces actions based on the current state of the environment, and the environment produces new states and rewards depending on the agent's actions. The rewards are values that indicate the adequacy of the agent's actions to a final goal. A reinforcement learning framework attempts to optimize the

policy by maximizing the expected sum of rewards produced by the environment under the actions generated by the policy. The iterative process between the environment and the agent (policy) is exercised with the objective of optimizing the expected total value of rewards in all successive steps.

This framework is originally intended for interactive agent-environment scenarios (**online scenario**) [8], but can be extended to supervised learning problems with labeled datasets (**offline scenario**) [7] by interpreting the actions as predicted labels, the state of the environment as predictor variables, and the environment as a sampling function that randomly selects samples from the labeled dataset. In this approach, the function that implements the environment is also responsible for generating rewards as values related to the prediction quality of the labels. The inclusion of the RBFNN as the policy network of an offline reinforcement learning framework assumes and further extends this approach.

The offline reinforcement learning scenario is important for applications where rewards cannot be directly assigned by the environment without human intervention, which means that we have to rely on labeled datasets to score the actions based on closeness between predictions and ground truth values (stored in the labeled dataset). NID corresponds to this type of application.

We tested the applicability of the proposed methods for various NID scenarios; however, the proposed methods are generic and perfectly applicable to implement classifiers in other fields. We have chosen NID for its complexity and strong limitations in terms of noisy and unbalanced data sets, which also appear in other fields (e.g., medicine, malware detection, agriculture. . .), and which could also benefit from these results.

We present a comprehensive analysis of results based on several performance metrics (Accuracy, F1-score, Precision and Recall) where the proposed extensions to the RBFNN architecture are applied to several distinctive intrusion detection (ID) datasets: AWID, UNSW-NB15, NSL-KDD, CICIDS2017 and CICDDOS2019. These are five of the most representative ID datasets [1]. The datasets exhibits quite different behaviors, allowing a better generalization of the conclusions. The five datasets correspond to five different NID scenarios, according to the types and distribution of intrusions. The proposed new models are compared to a broad set of alternative machine learning (ML) and deep learning (DL) models for all five datasets. The results obtained show that the new RBFNN extended architectures achieve the best results in most of the performance metrics for the five proposed scenarios.

Some interesting conclusions from this work, when the proposed methods are applied to NID, are: (a) The inclusion of additional dense layers in RBFNN networks has a significant impact on the classification performance. (b) Equally important is the selected loss function and learning framework for these networks. In particular, the proposed offline reinforcement learning scheme provides the best results, especially in the case of highly unbalanced and/or noisy datasets, which are common in network intrusion detection.

TABLE 2. Comparative summary between proposed RBFNN-based methods and similar alternative methods.

Models	End-to-end Learning	RBF Parameters Learning	Weight Parameters Learning	Loss Functions
Existing RBFNN	No	PSO, QPO, IRBF or SGD	Linear Regression (analytical) or SGD	Cross-entropy (CE) when using SGD
Proposed RBFNN	Yes	SGD	SGD	CE, Hinge loss or based on DRL

As a summary, Table 2 provides a comparison between the proposed RBFNN-based methods and alternative proposals based on similar models.

The main **contributions** of this research are to propose novel extensions of the RBFNN architecture, with the following objectives: (a) Propose a novel method for optimization of parameters/weights within a reinforcement learning framework, where the RBFNN classifier plays the role of the policy network in a Deep Reinforcement Learning (DRL) training scheme. (b) Adapt the reinforcement learning scheme from the usual online interactive agent-environment scenario to an offline scenario working with a labeled dataset (offline reinforcement learning) [7], [9]. (c) Provide an end-to-end training process where the weights, centers, and dispersion parameters of an RBFNN are learned simultaneously by gradient descent. (d) Propose extensions to the classic RBFNN configuration from a single hidden layer network to a multilayer network. (e) Show that the different RBFNN variants resulting from these new premises produce classifiers that can be especially useful for NID, considering the results obtained when applied to five different and representative ID datasets

This paper is organized as follows: Section II presents the works related to this research following different criteria. The data used for the experiments and the proposed algorithms are presented in Section III. The results of the different experiments are given in Section IV, with the final conclusions in Section V.

II. RELATED WORKS

There is a significant literature that presents solutions for network intrusion detection based on RBFNN. The RBFNNs introduced in the literature are usually not based on end-to-end training of the network using gradient descent, but on alternative optimization methods, e.g., linear regression, genetic optimization. These related works mainly use simple networks with a single hidden layer (the RBF layer itself), and typically a two-stage model optimization is applied, training the RBF parameters and the neural network weights

separately. Meanwhile, our proposed extension to RBFNN produces multi-layer models, allowing end-to-end training of the parameters of the entire network and with the ability to be flexible in the learning framework chosen (either supervised or reinforcement learning). In particular, our proposed architecture is, to our knowledge, the *first offline reinforcement learning architecture with an RBFNN* and, consequently, the first application of this model to NID.

The related works on RBFNN can be grouped into the following categories oriented by goals and methods used:

(a) **Application of regular RBFNN to NID with different network optimization methods:** Authors in [10] use the NSL-KDD dataset for intrusion detection by applying an RBFNN. They optimize the network parameters without using end-to-end training and gradient descent, but an algebraic solution based on the calculation of the pseudo-inverse matrix. They report high accuracy on an ad-hoc test set of 40000 samples that does not correspond to the proposed test set of 22544 samples provided by NSL-KDD. The work in 11] proposes an RBFNN where an optimal network structure and weights are obtained with a combination of quantum-behaved particle swarm optimization (QPSO) and gradient descent applied to the QPSO algorithm. An overview of the challenges for RBFNN-based intrusion detection systems is provided in [12], identifying as the most important aspects the optimization of the location of cluster centers and width spread for the RBF neurons and the optimization of the network weights. This literature review presents the recursive least mean squares method as the main proposal to adjust the weights, and various optimization algorithms to optimize the other RBF parameters, such as: IRBF, PSO and QPSO. None of the works in this review present an end-to-end learning approach based on gradient descent across the entire network. In [13] the authors use a simple RBFNN for intrusion detection in a host-based scenario with a proprietary dataset from the University of New Mexico. The model consists of a single hidden layer RBFNN where the output weights are optimized with linear regression with no gradient descent. They claim to obtain excellent detection results for this dataset. A similar approach to [13] is adopted in [14] for the KDD99 dataset, also reporting excellent results. The work in [15] presents RBFNN as an important asset to mitigate adversarial examples, applying the results to image datasets e.g. MNIST and CIFAR10. Authors in [16] propose an RBFNN with an optimization algorithm to obtain the cluster centers for the RBF network based on an immune optimization algorithm and a least squares recursive method to adjust the network weights. An alternative solution to calculate the RBF centers different from the usual k-means approach is offered in [17], using an eigenvector based clustering method; once the centers are obtained, the weights are optimized by gradient descent. There is no subsequent end-to-end training of the entire network. The work in [18] presents a simple one-hidden layer RBFNN to detect intrusions for the KDD99 dataset. They use Levenberg-Marquardt as the optimization algorithm, with an ad-hoc method to determine the RBF centers. Authors in [19] present a particle swarm optimization algorithm (PSO) for the optimization of parameters of the RBF layer. The algorithm is applied to the KDD99 dataset. In [20] the imperialist competitive algorithm is proposed to perform network optimization in an architecture that combines RBFNN with a self-organizing map.

(b) **RBFNN extensions:** In [21] is introduced a novel architecture for intrusion detection with kernel principal component analysis for features dimensionality reduction and an extreme learning machine model (ELM). It is applied to the KDD99 and UNSW-NB15 datasets with state-of-the-art (SOTA) results, achieving an average F1-score for the 10 labels prediction (UNSW-NB15) of 0.604. A different approach is adopted in [22] with a combination of RBF and Elman recurrent neural network which is appropriate for data with a time-series structure. The Elman layer is added as an additional layer after the RBF layer. The work in 23] proposes an iterative learning process that allows incorporating new intrusion labels by comparing the maximum value associated with the already expected labels with a threshold per label; if a sample provides output values smaller than the threshold it implies a new label and the addition of a new RBF neuron. The allocation of initial center values is done with an unsupervised self-organizing map algorithm. The method is applied to the KDD99 intrusion detection dataset. Authors in [24] use a neural network formed by an initial stage of stacked autoencoders to perform dimensionality reduction followed by an RBF layer. They apply the method to the UNSW-NB15 dataset for the 2-labels scenario using the MATLAB platform and an unclear procedure to perform RBF parameters optimization. Training the RBFNN network end-to-end using gradient descent is presented theoretically in 25] under a proposal for regularization theory and regularization networks.

(c) **Online reinforcement learning:** Applications of online reinforcement learning with RBFNN networks are mainly in the control engineering field. A Deep Q-Learning (DQN) reinforcement learning model is used in [26] with a RBF neural network as the policy network. The network weights are learned with gradient descent, but the RBF parameters are learned separately. The method is applied in the control field. A temporal difference reinforcement learning approach is applied in [27] with an RBFNN where their parameters are learned independently of the weights. The application is for a scheduling optimization problem. In [28], to solve a maze problem is used a DQN model

with an RBFNN with their parameters learned with an evolutionary algorithm. The work in 29] applies a RBF layer at the end of the network and all parameters are adjusted by gradient descent. It extends the DQN model to continuous control. The model is applied to a continuous state space for control applications. Likewise, in [30] the authors use gradient descent and particle swarm optimization (PSO) for the learning of parameters of a neural network used with an actor-critic reinforcement learning framework. It is applied to an adaptive controller implementation. This line of work is also applied in 31]–[33] with an actor-critic DRL model applied to a continuous state-space applied to control problems. In this case the RBF parameters are adjusted as part of the training. In all these works, the DRL framework is applied to an online optimization/control problem. There are also interesting online learning works based on autoencoders [34], [35].

(d) **Offline reinforcement learning**: The application of deep reinforcement learning (DRL) to NID is a topic of increasing interest [36]. DRL is generally limited to online environments modeled by a Markov decision process corresponding to an interactive environment. This scheme cannot work with a dataset of recorded attacks, which is the subject of offline reinforcement learning [9]. There are currently few applications of offline reinforcement learning for NID [7], [37] even though it is currently an important area of research as an extension/alternative to supervised learning schemes [9].

Table 3 presents a summary of the related works offering a comparison between their main characteristics. The two groups of proposed solutions, with their main characteristics, are also included at the end of the table, to help compare the alternative works and the differences with the proposed methods.

TABLE 3. Categorization of related works with their main distinctive characteristics, and comparison with the characteristics of the proposed solutions.

	Work	Characteristics	Dataset
RBFNN classic	[10]	Linear regression (analytical solution)	NSL-KDD
	[11]	QPSO plus gradient descent	KDD99
	[13]	Linear regression (analytical solution)	Proprietary
	[14]	Linear regression (analytical solution)	KDD99
	[15]	Random initialization of parameters plus gradient descent	MNIST, CIFAR
	[16]	Immune optimization algorithm and least squares	KDD99
	[17]	Eigenvector based clustering method plus gradient descent	UC Irvine ML Repository
	[18]	Levenberg-Marquardt weight optimization, with ad-hoc method for RBF centers	KDD99
	[19]	PSO plus linear regression	KDD99
	[20]	Imperialist competitive algorithm plus SOM	KDD99
RBFNN extensions	[21]	Kernel PCA for dimensionality reduction plus ELM	KDD99, UNSW-NB15
	[22]	RBF layer and Elman recurrent neural network	1999 DARPA ID Evaluation
	[23]	Unsupervised SOM plus linear regression	KDD99
	[24]	Stacked autoencoders plus RBF layer	UNSW-NB15
	[26]	DQN with nonintegrated optimization: k-means plus gradient descent.	Online control
Online RL	[27]	Temporal difference with nonintegrated optimization: k-means plus gradient descent.	Scheduling optimization
	[28]	DQN with nonintegrated optimization: evolutionary plus gradient descent.	Path optimization
	[29]	DQN with gradient descent	Continuous control
	[30]–[33]	Actor-critic with nonintegrated optimization: PSO plus gradient descent.	Continuous control
Offline RL	[7]	DQN, DDQN, actor-critic, policy gradient without RBFNN	NSL-KDD, AWID
	[37]	DDQN, Dueling DRL, actor-critic without RBFNN	NSL-KDD, AWID
Extended RBFNN	This work	All the parameters (Weights, Centers and Dispersion) are learned jointly end-to-end with gradient descent. Cross-entropy and Hinge losses.	NSL-KDD, UNSW-NB15, AWID, CICIDS2017, CICDDOS2019
Extended DRL+RBFNN	This work	DDQN with RBFNN as policy network, with all the parameters learned jointly end-to-end with gradient descent.	NSL-KDD, UNSW-NB15, AWID, CICIDS2017, CICDDOS2019

III. WORK DESCRIPTION

This Section describes (a) the datasets used to perform the comparison between the models proposed in this work, and (b) the new proposed models in detail. The datasets applied are introduced in Section III.A. The different proposed models are described in Section III.B.

A. SELECTED DATASETS

To explore the different challenges imposed by an intrusion detection problem [1], which is characterized by complex behavior patterns with unbalanced datasets, we have used five ID datasets with distinctive properties to draw better conclusions on the prediction performance of the proposed models.

We will use the NSL-KDD and UNSW-NB15 datasets for binary classification and the AWID, CICIDS2017 and CICDDOS2019 datasets for multiclass classification. The two-label scenario for NSL-KDD provides a reasonable balanced dataset obtained from an underlying very different frequency distribution of basic attacks between the training and test sets. Additionally, it presents a change in the distribution of the majority class between the Normal and Attack classes for the training and test sets. This change in the distribution of the majority class is reversed for the UNSW-NB15 dataset for the two-label scenario. UNSW-NB15 is also very

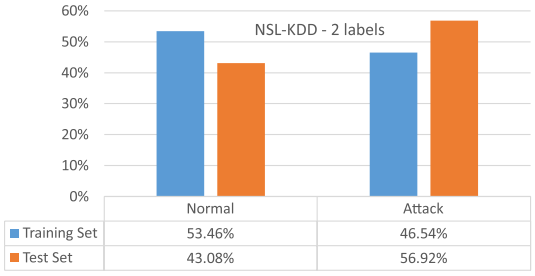

FIGURE 1. Frequency of intrusions for the training and test datasets (NSL-KDD) for the 2-labels scenario.

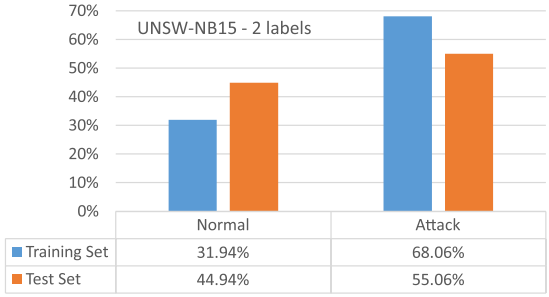

FIGURE 2. Frequency of intrusions for the training and test datasets (UNSW-NB15) for the 2-labels scenario.

different from NSL-KDD in having the same basic attacks for the training and test sets with a very similar frequency distribution between them, which is far from being the case for the NSL-KDD dataset. The AWID dataset is the most unbalanced one with four basic attacks and has been chosen to verify the behavior of the different models for a multiclass classification. The CICIDS2017 and CICDDOS2019 datasets also have four unbalanced classes, but not as unbalanced as AWID.

Each dataset (NSL-KDD, UNSW-NB15, AWID, CICIDS2017 and CICDDOS2019) with the chosen label configuration (2, 2, 4, 4 and 4, respectively) provides different characteristics, restrictions and challenges allowing a better validation of results (Section IV).

1) NSL-KDD DATASET

The NSL-KDD dataset is an intrusion detection dataset evolved from the original KDD-99 [38] dataset. The NSL-KDD dataset is considered classic, but it remains a reference for intrusion detection problems as it is extremely well-known with a large literature that uses it. The NSL-KDD dataset is composed of 125,973 and 22,544 training and test samples, respectively. It contains 41 features, with 38 continuous and 3 categorical. The continuous features have been scaled to the [0–1] range. The categorical features are one-hot encoded. The resulting dataset has 122 features with 38 continuous and 84 (one-hot encoded) binary features. The features are formed by aggregating the information contained in the data packets associated to the normal/attack traffic.

The total number of distinct labels is 40. The number of labels presented in the training and test datasets is different with 23 in the training and 38 in the test sets. There are labels in the test set which are not in the training set and vice versa (16,6% of the test samples correspond to a class that is not present in the training set); this contributes to creating a particularly noisy dataset. The original labels can be grouped according to various hierarchies into different categories [38]. Fig. 1 presents the frequency distribution of the labels grouped into two categories: Normal and Attack, for the training and test sets. In this work we will use this grouping into two distinct categories.

2) UNSW-NB15 DATASET

The UNSW-NB15 [39] is an intrusion detection dataset. It is a more recent and larger dataset than NSL-KDD, with 2,540,044 and 82,332 training and test samples, respectively. The dataset provides 10 labels (9 attacks plus normal traffic) with 42 features: 39 continuous and 3 categorical. Following a preprocessing similar to that presented for NSL-KDD (Section III.A.1), the total number of final features is 196. Similar to NSL-KDD, the features are formed by aggregating the information of the data packets.

It is an imbalanced dataset. Fig. 2 presents the frequency distribution of the labels grouped into two categories: Normal and Attack, for the training and test sets. In this work we will use this grouping into two distinct categories. The label distribution for the training and test sets is quite similar to NSL-KDD, but with the majority class swapped in the training set (i.e., Normal is the majority class for the NSL-KDD training set while Attack is the majority class for the UNSW-NB15 training set).

3) AWID DATASET

Aegean Wi-Fi Intrusion Dataset (AWID) [40] is a public intrusion detection dataset with three types of attacks for the IEEE 802.11 networks. It is also a larger and more up-to-date dataset than NSL-KDD. AWID provides several dataset instances from which the AWID-CLS-R has been chosen. This dataset has four classification labels: normal, flooding, injection, and impersonation. It provides 1,795,574 and 575,642 training and test samples, respectively. The dataset has a total of 154 continuous and categorical features. The number of features can be reduced to 24 after removing features with less importance and with null and constant values [40], which after a preprocessing similar to NSL-KDD (Section III.A.1) produces a final feature number of 58.

It is an imbalanced dataset with frequencies from 91% to 9% for the Normal and Attack labels. It is more unbalanced than the NSK-KDD and UNSW-NB15 datasets, but contrasting with NSL-KDD, the label frequencies for the training and test sets are extremely similar, more similar than the corresponding frequency distribution in UNSW-NB15. Fig. 3 presents the frequency distribution of all distinct labels for the training and test sets. In this work we will use the original

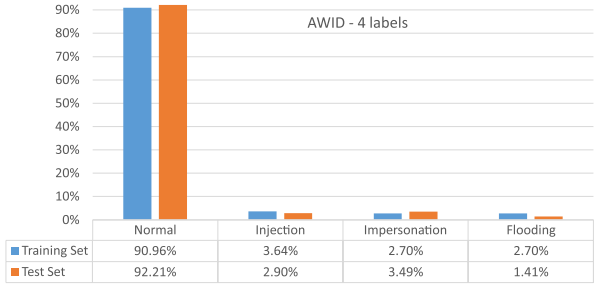

FIGURE 3. Frequency of intrusions for the training and test datasets (AWID) for the 4-labels scenario.

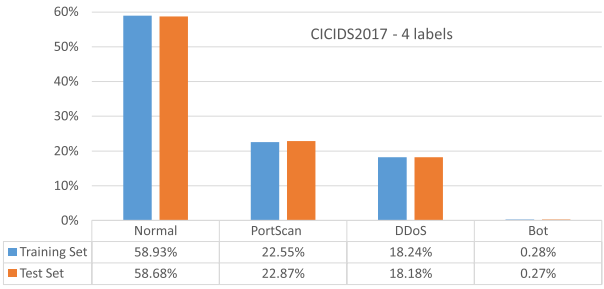

FIGURE 4. Frequency of intrusions for the training and test datasets (CICIDS2017) for the 4-labels scenario.

four labels to perform multiclass classification, which will allow us to test the algorithms under a particularly unbalanced dataset.

4) CICIDS2017 AND CICDDOS2019 DATASETS

CICIDS2017 [41] and CICDDOS2019 [42] datasets are two recent intrusion detection datasets from University of New Brunswick (UNB), Canada. CICIDS2017 contains three types of attacks plus normal traffic. CICDDOS2019 is dedicated to distributed denial of services (DDoS) attacks, also containing three types of DDoS attacks plus normal traffic. CICIDS2017 has 70 continuous and 7 categorical features with 562539 training and 139670 test samples. CICDDOS2019 has 67 continuous and 7 categorical features with 539579 training and 134884 test samples. The features have been processed, eliminating the features with very low variance, scaling the continuous features in the range of values [0-1], and one-hot-encoding the categorical ones.

Figs. 4 and 5 present the frequency distribution of all distinct labels for the training and test sets of these datasets.

CICIDS2017 and CICDDOS2019 have been selected for being modern multi-class NID data sets with different types of attacks. They are unbalanced but not as extremely unbalanced as AWID, and offer the opportunity to explore other types of prediction behaviors for the proposed classifiers.

B. MODEL DESCRIPTION

We propose a novel extension of the RBFNN basic model. The RBFNN basic architecture is based on a single hidden-layer where each neuron implements a radial basis

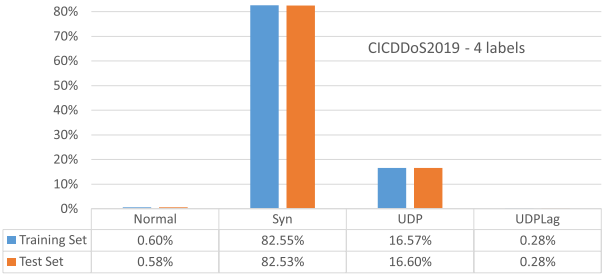

FIGURE 5. Frequency of intrusions for the training and test datasets (CICDDOS2019) for the 4-labels scenario.

activation function e.g., Gaussian. These functions are also called radial kernels, and are defined by two parameters: a center and a scaling/dispersion parameter [6]. These parameters are usually optimized (learned from data) separately and with ad-hoc techniques, such as: clustering techniques (e.g., k-means) or generic optimization methods (e.g., PSO, Quantum PSO). The weights between the RBF layer and the output layer are also separately optimized using least squares or gradient descent. Therefore, the learning process for the basic RBFNN consists of optimizing three types of parameters (centers, dispersion/scaling, and weights) within three distinct and separate learning tasks [6].

In this work, an end-to-end learning process is proposed in which the three types of parameters are learned simultaneously end-to-end by gradient descent. We consider the centers and dispersion parameters as additional special weights that are included in the computational graph of the network [43]. This end-to-end and simultaneous learning of the complete set of parameters allows us to consider: (a) additional loss functions (e.g., hinge-loss for maximum class separation), (b) additional layers (and layer configuration) to be incorporated after the first RBF layer, and (c) to include the entire RBFNN network as the policy function of a DRL model, where the differentiable loss function is replaced by a more generic and not necessarily differentiable reward function [7].

The extended RBFNN models described here are divided into two groups according to: (a) using a supervised learning framework based on direct minimization of the classification errors (Fig. 6), or (b) using an offline reinforcement learning framework to train the RBFNN model (Fig. 8). In both cases, all model parameters are learned end-to-end by gradient descent.

1) EXTENDED RBFNN WITHIN A SUPERVISED LEARNING FRAMEWORK

Fig. 6 shows the proposed RBFNN architecture where the first hidden layer is always a layer with neurons that implement a radial basis activation function. In Fig. 6, the RBF layer is shown in detail by zooming into an RBF neuron. The RBF activation (1) is different to a sigmoid or ReLU activation. The dot product between the input and the weights, appropriate for a sigmoid or ReLU activation, is replaced in the case of an RBF neuron by the difference between the input

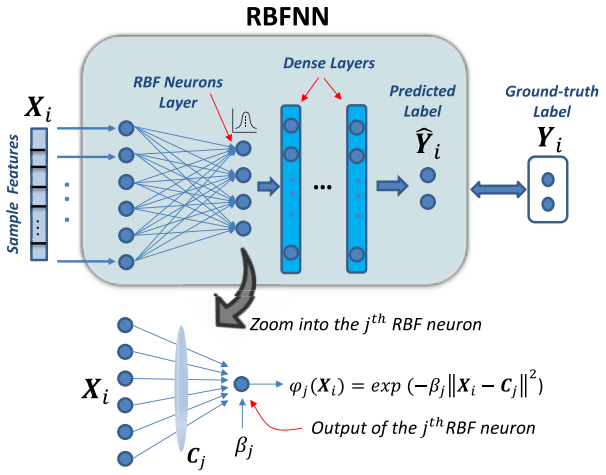

FIGURE 6. Generic architecture of the extended RBFNN within a supervised learning framework.

and the RBF center. The RBF activation acts on this difference by computing its squared norm followed by a scaled exponential function instead of a sigmoid or ReLU function. The resulting expression is differentiable and feasible to incorporate into an end-to-end gradient descent algorithm.

A radial basis function (RBF) is a real-valued function whose output depends on the distance between the input and a certain fixed point that we will call the center. The center values have similar dimension to the input values. For this study we will use a particular type of RBF: Gaussian RBF, which has the form shown in (1), where $\boldsymbol{X}$ is the input vector, β is a scaling/dispersion parameter (scalar value), $\boldsymbol{C}$ is the centers vector and $\|\ldots\|^2$ is the squared vector norm, which in our case will be the squared Euclidean distance [6]:

$$\varphi(\boldsymbol{X}) = \exp(-\beta \|\boldsymbol{X} - \boldsymbol{C}\|^2) \tag{1}$$

Note: In mathematical expressions a term in bold indicates a vector.

Each neuron implementing the activation function in (1) will have a distinct center and dispersion values; these values are considered fixed after training, but will be adjusted during training to achieve an optimal value that minimizes the defined loss function. The first layer should always be the RBF layer since the intention is to apply the different RBF functions (one per RBF neuron) to the input, i.e., each RBF neuron will provide a value related to the distance from the input to its center ($\boldsymbol{C}$), where the distance is scaled by the scaling/dispersion parameter (β). A graphical representation of how the RBF layer works is provided in Fig. 7, for an RBF layer with three neurons, where $\boldsymbol{X_i}$ is a particular input sample and $\boldsymbol{C_j}$ and β_j are, respectively, the center and dispersion parameter for the RBF neuron j.

After the RBF layer, the architecture may have several subsequent layers, which may be generic in nature, but, for this work, we have considered fully connected (FC) dense layers (Fig. 6). The final layer will provide the prediction ($\hat{\boldsymbol{Y}}$) which can be generated with: 1) a softmax activation that offers normalized probabilities, which are interpreted as the

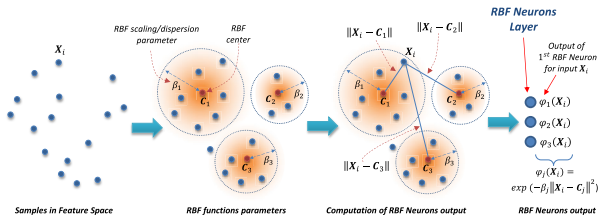

FIGURE 7. Graphical depict on how an RBF layer works. The diagram shows an RBF layer with three neurons acting on an input sample ($\boldsymbol{X_i}$), where the output of neuron j (i.e., φ_j) is computed with an RBF function (1) with particular center ($\boldsymbol{C_j}$) and dispersion (β_j) parameters.

probability of the associated class; or 2) a linear activation that produces real values, and where the selected class is associated with the highest output value. Related to the type of output produced, we can employ different types of loss functions. In this work we apply two loss functions: cross-entropy (2), and hinge-loss (3):

$$Cross\ Entropy\left(\boldsymbol{Y}, \hat{\boldsymbol{Y}}\right) = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=1}^{L}\left[Y_{i,j} log\left(\hat{\boldsymbol{Y}}_{i,j}\right) + \left(1 - Y_{i,j}\right) log\left(1 - \hat{\boldsymbol{Y}}_{i,j}\right)\right] \tag{2}$$

$$Hinge\ Loss\left(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, \Delta\right) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j \neq z_i}\left[\max(0, \hat{\boldsymbol{Y}}_{i,j} - \hat{\boldsymbol{Y}}_{i,z_i} + \Delta\right] + \gamma \|\boldsymbol{W}\|^2 \tag{3}$$

In (2), the cross-entropy between two sets of N predicted values ($\hat{\boldsymbol{Y}}$) and their corresponding ground-truth values ($\boldsymbol{Y}$) is an average of the log probability of the predicted class, where L is the number of predicted classes, and $Y_{i,j}$ is a {0,1} value associated to the one-hot encoded ground-truth value for class j of sample i. In this case, $\hat{\boldsymbol{Y}}_{i,j}$ corresponds to the associated predicted value for class j of sample i, which is a value in the range [0, 1] and interpreted as the probability that sample i (i.e., $\boldsymbol{X_i}$) belongs to class j.

In (3), the hinge-loss is also defined as the average of a distance between two sets of N predicted values ($\hat{\boldsymbol{Y}}$) and their corresponding ground-truth values ($\boldsymbol{Y}$). In this case, the distance is a real value distance (for each sample i), between the output for each class other than the correct class (indexed by z_i) if the distance is greater than a margin (Δ), and zero otherwise. In this case, $\hat{\boldsymbol{Y}}_{i,j}$ corresponds to a real value (not a probability), and the predicted class for sample i (i.e., $\boldsymbol{X_i}$) is chosen by selecting the class j that maximizes $\hat{\boldsymbol{Y}}_{i,j}$. The cross-entropy loss requires the use of a softmax activation for the last layer and the hinge-loss a linear activation. In (3) (hinge-loss) the term $\|\boldsymbol{W}\|^2$ corresponds to the norm of the weight vector (sum of the squares of its values) and is a regularization term to avoid over-fitting, with the parameter γ adjusting the effect of this regularization term.

Previous to the final adjustment by gradient descent, and, instead of a random initialization, we use an initialization of the RBF centers using k-means and a fixed value for the dispersion parameter (normally a value of 2). Some more complex alternative methods for centers initialization are: (a) Learning Vector Quantization (LVQ) [44], (b) RBF

center selection based on Fisher Ratio Class Separability measure [45], and (c) PCA based k-means, which is a novel method that could be of interest for large datasets [17]. It is important to note that this is just an initialization of parameters that will be further adjusted by gradient descent during training; therefore, in our case, it is not necessary to use complex methods, the simple and fast k-means algorithm is sufficient to provide an effective initialization that improves the convergence and the time required for the learning process.

To train the RBFNN model within a supervised learning framework (Fig. 6), we have used 100 epochs and a batch size of 20 to 200 samples (depending on the dataset). We have used early-stopping to avoid overfitting with a patience period of 10 and using a validation set of 20% of the training set [43].

2) EXTENDED RBFNN WITHIN AN OFFLINE REINFORCEMENT LEARNING FRAMEWORK

An alternative proposal consists of extending the training of an RBFNN from a regular supervised scheme to an offline reinforcement learning scheme, where the RBFNN plays the role of the policy network interacting with a specifically designed 'environment' (Fig. 8). As presented in the Introduction, a reinforcement learning framework contains two elements: an agent (implementing a policy) and an environment. The policy produces actions based on the current state of the environment, and the environment produces new states and rewards depending on the agent's actions. The rewards are values that indicate the adequacy of the agent's actions to a final goal. The objective is to optimize the policy by maximizing the expected sum of rewards produced by the environment under the actions generated by the policy. The policy can be implemented through an intermediate function that gives a value for the quality (Q function) of each state-action pair, and by choosing the action that maximizes the Q function we can select the best action for each state produced by the environment, hence implementing the policy. The Q function can be parameterized, and implemented with a neural network (NN). This NN can be trained end-to-end with a loss function that attempts to reduce the error between the output of the network and the estimation of the sum of future rewards that depends on the Q function (Bellman's equation) [8]. Using a reinforcement learning scheme, the RBFNN parameters can be updated, during training, using one of several deep reinforcement learning approaches e.g. Deep Q-Learning (DQN), Double DQN (DDQN), Policy gradient, Actor-critic [46].

The offline reinforcement learning adopted is outlined in Fig. 8, where a specifically designed environment generates new training samples by random sampling the dataset and producing reward values based on the errors made by the policy network, which is implemented by the RBFNN (Fig. 8). This learning approach allows assimilating the loss function (supervised learning) to the reward generation process (reinforcement learning), and the space of possible actions to the

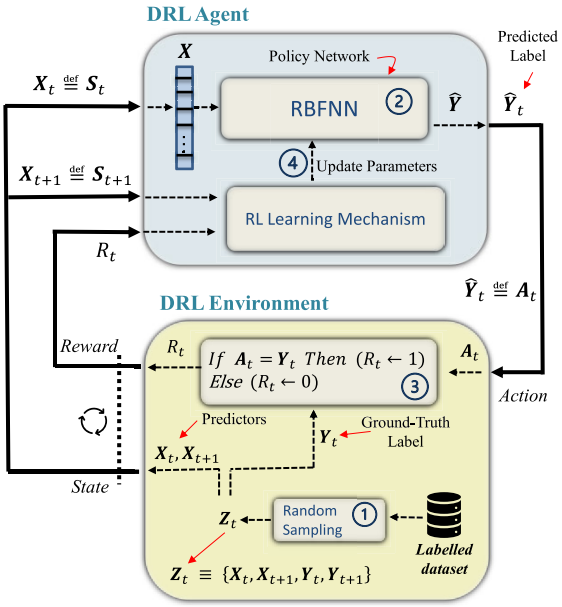

FIGURE 8. High-level view of the extended RBFNN playing the role of the Policy Network in an offline reinforcement learning framework. The circled numbers indicate the process steps following the algorithm in Fig. 9. The symbol $\leftarrow$ acts as an assignment operator. The operator $\stackrel{def}{=}$ means that their left and right operands are defined as being equal. The whole process is repeated multiple times indicated by the circular loop symbol (to the left of the DRL environment).

labels to be predicted by the algorithm. The reward function is open to any function that associates a higher value with better predictions. In this work we have used the DDQN [47] as our DRL algorithm, and a simple 1/0 reward function (1 in case of positive reward and 0 otherwise), but other DRL algorithms and reward functions are possible. Fig. 8 provides a high-level view of the process involved, where 4 steps can be observed in each learning cycle. Each learning cycle is iterated multiple times until the learning process is complete (indicated by the circular loop symbol to the left of the DRL Environment). The convergence of the learning process is discussed in Section IV.G.

Fig. 9 provides a more detailed view of the offline DRL training algorithm presented in Fig. 8. The training process is based on a cycle with 4 basic steps, shown as circled numbers in Fig. 8 which correspond to the numerical sequence in Fig. 9. The algorithm in Fig. 9 corresponds to an adapted Q-learning algorithm [8] that allows creating an intrusion detection classifier using a labeled dataset. The Q-learning algorithm attempts to find the best Q-function for the agent (the RBFNN model). A Q-function receives as input a state-action pair and generates a score value that indicates how good it is to take that action for that state. The Q-function can be obtained iteratively by (4) (Bellman's equation) [8]; where $\boldsymbol{S}_t$ is the current state, $\boldsymbol{S}_{t+1}$ is the next state, $\boldsymbol{A}_t$ is the current action, R_t is the current reward value.

The parameters α and λ are two hyperparameters: learning rate and discount factor, respectively. The future Q-value (Q_{t+1}) is formed by updating the current value (Q_t) with a weighted sum of the current reward (R_t) plus the difference

between the discounted maximum Q-value for the next state (choosing the best future action) ($\max_{A} Q_t(S_{t+1}, A)$) with the current Q-value ($Q_t(S_t, A_t)$). The learning rate is established by the optimization function used to train the NN and, in our case, this parameter is dynamically adjusted using the Adam optimization algorithm [48]. The parameter λ defines the importance of future rewards and is set low in our case because the next state is uncorrelated with the current state (due to random sampling of the labeled dataset), and the goal is to achieve a correct prediction for the current state i.e., a high current reward:

$$Q_{t+1}(S_t, A_t) \leftarrow Q_t(S_t, A_t) + \Phi \quad (4)$$

$$\text{where: } \Phi = \alpha \left[R_t + \lambda \max_{A} Q_t(S_{t+1}, A) - Q_t(S_t, A_t) \right] \quad (5)$$

In the above equation (and the rest of the paper) the symbol $\leftarrow$ acts as an assignment/update operator. As seen in Fig. 8 and Fig. 9, in this work, we have defined a correspondence between the state S_t with the features X_t extracted from the samples of the dataset, and the action A_t with the predicted label $\hat{Y}_t$, where the ground-truth label is represented as Y_t.

The Q-function is provided within the Policy Network which is implemented by the extended RBFNN architecture (Fig. 6). The input to the RBFNN is the current state, associated to the features from the labeled dataset, and the output represents the Q-function for the set of available actions, where each action corresponds to the selection of a possible intrusion label. The Q-function corresponds to how good it is to take some particular action in the current state. The action that, for the current state, provides the highest Q-value will be chosen as the predicted action (predicted label).

The algorithm used to train the extended RBFNN architecture within a reinforcement learning framework, is presented in Fig. 9. The algorithm in Fig. 9 begins with a random initialization of the weights of the RBFNN network that implements the Q-function ($Q_{t=0}$), and setting the discount factor (λ) to a low value. The algorithm has two loops: one to iterate through the number of epochs and another inner loop that iterates through the number of batches within each epoch. At the beginning of each epoch-loop there is a shuffle of the dataset to ensure subsequent random sampling. The 4 steps within the batch-loop are:

1) Load samples from the dataset (X_t, Y_t, X_{t+1}, Y_{t+1}).
2) Obtain the prediction $\hat{Y}_t$ made by the RBFNN (the policy) for the current X_t. This prediction corresponds to the action that maximizes the Q-function for the current state ($arg\max_{A} Q_t(S_t, A)$).
3) Assign a reward to the error between the predicted label ($\hat{Y}_t$) and the ground-truth label (Y_t). We have chosen a simple reward where an error is associated with a reward of 0 and a correct prediction with a regard of 1. Other rewards have been analyzed [7] but this simple one has provided the best results.
4) Calculate a reference Q-value, given by $Qref_t$; where, $Qref_t = R_t + \lambda \max_{A} Q_t(S_{t+1}, A)$, formed by the current reward (R_t) and the discounted maximum Q-value achievable for the next state with the best possible action.

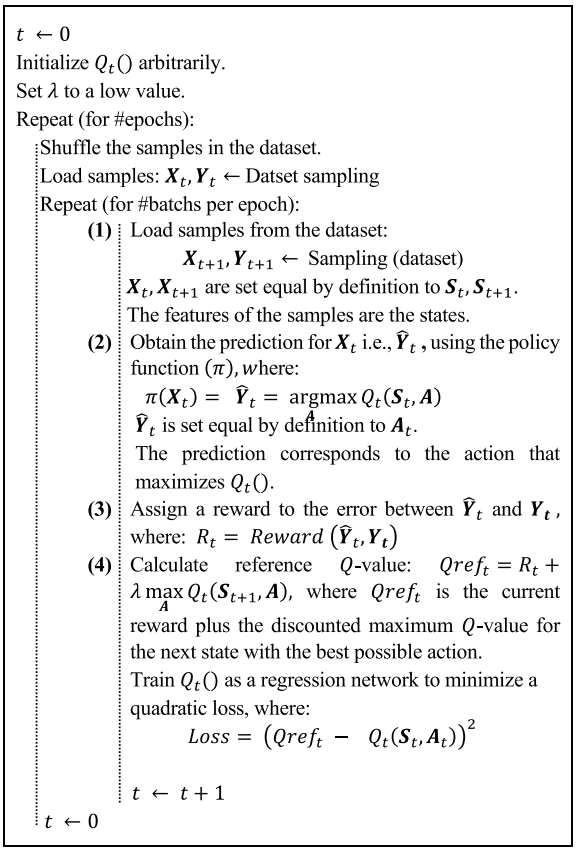

FIGURE 9. Algorithm to train the extended RBFNN architecture working as the Policy Network of a reinforcement learning framework. The 4-step sequence of the inner-loop corresponds to the circled numbers in Fig. 8. The symbol $\leftarrow$ acts as an assignment operator.

It is important to realize that the Q-value for the next state is calculated with the current Q-function (Q_t). The value of $Qref$ is used to perform a batch training of the RBFNN as a regression network minimizing a quadratic loss, where the loss is: $(Qref_t - Q_t(S_t, A_t))^2$, that is, we try to minimize the difference between the current Q-value ($Q_t(S_t, A_t)$) with the reference one ($Qref_t$). This batch-step training will perform a weights update of the RBFNN producing a new version of the Q-function (Q_{t+1}) according to (4).

It is interesting to compare the proposed offline DRL algorithm (Fig. 8) with the original online DRL algorithm based on the DQN framework [49] which is depicted in Fig. 10. The learning process of an online DQN framework (Fig. 10) is done in two basic steps. In step 1, the policy network (implemented with a neural network) interacts with the environment several times. This iteration produces a sequence of states, rewards and actions. This sequence is organized as a sequence of tuples, with four elements per tuple: (S_t, A_t, S_{t+1}, R_t). After a number of iterations, this sequence is stored and used in step 2 of the learning process. In step 2, the stored tuples are randomly sampled (called experience replay) and used to update the Q function in a process similar to the outlined in

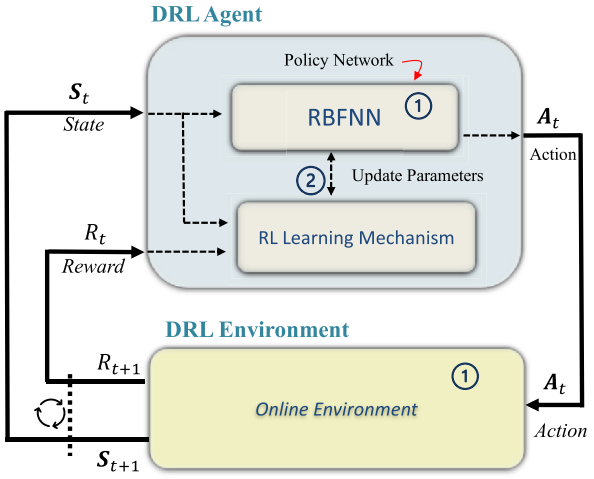

FIGURE 10. High-level view of the extended RBFNN playing the role of the Policy Network in an online reinforcement learning framework. In this case, the learning process has essentially two steps. In step 1, the policy network interacts with the environment multiple times. The result of this iteration is a sequence of samples each formed by a tuple: (S_t, A_t, S_{t+1}, R_t). In step 2, the sequence of previously gathered and stored samples is used to update the policy network through the Q function. The whole process is repeated multiple times indicated by the circular loop symbol (to the left of the DRL environment).

Fig. 9 [49]. After step 2, the policy network is updated (via the Q function), and the new updated policy network will be used to interact with the environment in the next round of the process (in the following step 1). Steps 1 and 2 are repeated multiple times, until the learning process is complete (indicated by the circular loop symbol to the left of the DRL Environment). We can see that similar principles are used in both the online and offline versions of the algorithm, with the exception of the necessary adaptation of the environment (offline version) and a different organization of the training steps.

To train the RBFNN model with the offline DRL learning framework (Fig. 8), we have used 30 to 100 epochs and a batch size of 20 to 100 samples (depending on the dataset). In this case the activation function of the last layer is a linear function, and the training is performed as a regression network. We have included an initial exploration phase with an ε-greedy algorithm, where the best action produced by the policy is selected with probability p or a random action with probability 1-p. During the first 7 epochs (exploration phase), the action produced by the policy has been assigned by an annealed ε-greedy to a random label with a probability ranging from 10% to 0.1%, with a smooth linear reduction of this probability between its maximum and minimum values during the exploration phase. After the initial exploration phase, the label produced by the policy is always the one with the highest output value (best action).

IV. RESULTS

In this Section, we apply several representative machine learning (ML) classification models to the five datasets presented in Section III.A (NSL-KDD, UNSW-NB15, AWID, CICIDS2017 and CICDDOS2019) in order to compare their prediction performance results with the ones obtained by the proposed new models based on RBFNN (Section III.B). The ML models used for comparison are similar for all datasets providing a homogeneous test benchmark. All of these alternative ML methods and results are shown in Figs. 10-12 for each dataset, along with the results produced by the proposed methods under the RBFNN category. The alternative ML models used for comparison are some of the most representative models currently in use at NID [4], [50], [51]: (a) Multinomial Logistic Regression (LR), (b) Linear and RBF Kernel Support Vector Machine (SVM), (c) Random Forest (RF), (d) Gradient Boosting Machine (GBM) with tree stumps, (e) AdaBoost based on trees, (f) Neural Network (NN) with simple dense layers in a Multilayer Perceptron (MLP) configuration, (g) Convolutional Neural Network 1-Dimensional (CNN-1D) [52], and (h) Linear Model with a Kernel Approximation (LM+KA) [53]. The extended RBFNN models are divided into two groups according to the use of the DRL learning framework to train the RBFNN model (DRL+RBFNN) (Section III.B.2), or the use of a supervised learning framework based on direct minimization of the classification errors (RBFNN) (Section III.B.1); in any of these cases, all the RBFNN parameters (centers, dispersion and weights) are trained end-to-end and simultaneously. The NSL-KDD and UNSW-NB15 datasets will be used for binary classification and the AWID, CICIDS2017 and CICDDOS2019 datasets for multiclass classification. All results are obtained with the five selected datasets (Section III.A) using their original training and test sets without any reassignment or ad-hoc selection of training/test sets, which could provide better claimed results but make it difficult to perform homogeneous comparisons between results from different studies.

The classification metrics proposed are: Accuracy, F1-score, Precision and Recall. These performance metrics are based on the number of correct/incorrect predictions, separating true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), based on identifying an intrusion with a 'positive' case, and normal traffic with a 'negative' case. The above metrics are defined as follows:

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \tag{6}$$

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{7}$$

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{8}$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{9}$$

In the above equations, the symbol '#' represents the cardinality operator e.g., the total number of false negatives is represented as $\#FN$.

For this work, which deals with unbalanced datasets, we will consider the F1-score as the reference metric to rank the models. For the multi-class scenario (AWID, CICIDS2017 and CICDDOS2019 datasets) it is necessary to

aggregate the results for the different classes (label values), having different aggregate options: macro, micro, samples and weighted; opting for the weighted average as defined in [54]. The ROC-AUC metric, which is an important metric for binary classification, has not been used because some methods (e.g., SVM) have difficulties to generate a probability associated to the predicted label. Nevertheless, ROC-AUC has been used to compare different RBFNN models within this study (Fig. 16).

The results in Tables. 4-8 are color-coded with an array of colors from green to red, where the darkest green represents the best results and the darkest red the worst. The color code is applied column-wise, which allows to appreciate where the highest concentration of best results is found by category. All results for the RBFNN models are presented in Figs 10-12 in the RBFNN category (''Extended RBFNN models'') with the column: ''Model detail'', detailing the model configuration using common symbols with specific meaning and structure. Each model configuration is represented by the following common string: RBF(x)+yFC($z1$, $z2$...), v, w. Where x stands for the number of RBF neurons of the RBF layer; y stands for the number of subsequent fully connected (FC) layers, and when this number is one it means the RBF layer is the only hidden layer; the series $z1$, $z2$, ... stands for the number of neurons in each of the FC layers; v stands for the loss functions used which can be cross-entropy (CE), hinge-loss with a margin value of 1 (HS), hinge-loss with a margin value of 0 (HP) or a loss function obtained indirectly by using a DDQN deep reinforcement learning scheme (DDQN); and, w stands for the last layer activation function which can be linear (LI) or softmax (SM). The symbol [RND], following one of the $z1$, $z2$... series indicates that the corresponding layer has its weights set at random and are not trainable; in this case, the resulting model shares similarities with an Extreme Learning Machine (ELM) [55].

Tables 4-8 also include the required execution times for training and prediction (applied to the corresponding test set).

The results presented in Figs. 10-12 for the RBFNN category are for the most representative or best performing network configurations for each data set. It is important to mention that the binary classifiers for the NSL-KDD and the UNSW-NB15 datasets produce uncalibrated probability outputs. Either strongly biased towards zero (NSL-KDD) or one (UNSW-NB15), this is related to how the majority class of the training set is encoded, being the Normal class (class 0) for the NSL-KDD and the Attack class (class 1) for the UNSW-NB15. These uncalibrated results have been considered to establish the corresponding separation threshold between classes to achieve the best F1-score.

A. NSL-KDD RESULTS

The results for the NSL-KDD dataset are given in Table 4. We can observe how the best results (F1-score) are obtained for the DRL+RBFNN and RBFNN models with several subsequent FC layers, and how their width has the greatest effect on prediction performance. With the addition of a single layer we obtain excellent results, not improving much when adding additional layers. The best results are obtained for the cross-entropy loss function with a softmax activation for the last layer. The use of the hinge-loss does not provide good results and this behavior is also seen in the other datasets.

TABLE 4. Performance metrics for all models applied to the NSL-KDD dataset. Detailed results with column-wise color-coded metrics (from green to red, the greener the better). In addition, the two best values per column are marked in bold-italic.

Model			Accuracy	F1	Precision	Recall	Exec. Times (sec)	
Category		Model detail					Training	Test
Alternative ML models	LR	Logistic Regression	0.707	0.681	0.896	0.549	***86.4***	***0.03***
	SVM	Linear Kernel	0.774	0.772	0.907	0.672	***143.7***	***0.02***
		RBF Kernel	0.880	0.893	0.908	0.878	1696.2	158.65
	RF	Random Forest	0.747	0.721	***0.969***	0.574	194.6	3.87
	GBM	Gradient Tree Boosting	0.776	0.761	***0.969***	0.627	560.5	4.39
	AdaBoost	Adaboost with Trees	0.761	0.740	0.968	0.599	201.4	1.69
	NN	Neural Net (MLP), 3FC (200,50),CE,SM	0.797	0.788	0.968	0.665	601.0	0.61
	CNN	CNN-1D	0.806	0.799	0.969	0.680	1389.7	1.30
	LM+KA	Nystroem	0.887	0.894	0.956	0.840	533.6	0.55
Extended RBFNN models	RBFNN	RBF(40)+2FC (20),CE,SM	0.873	0.892	0.865	0.921	265.1	0.49
		RBF(60)+2FC (20),CE,SM	0.889	0.904	0.890	0.917	363.9	0.59
		RBF(80)+2FC (20),CE,SM	0.896	0.911	0.890	0.932	364.6	1.03
		RBF(80)+2FC (20),HS,LI	0.780	0.767	0.967	0.636	677.3	0.99
		RBF(80)+2FC (20),HP,LI	0.780	0.785	0.887	0.704	670.2	1.02
		RBF(80)+4FC (200,40,20),CE,SM	0.892	0.904	0.914	0.894	683.5	1.05
		RBF(80)+4FC (160[RND],40,20),CE,SM	***0.902***	0.915	0.901	0.929	400.5	1.12
	DRL+RBFNN	RBF(80)+4FC (150,100,20),DDQN,LI	0.899	***0.915***	0.871	***0.964***	1331.1	1.65
		RBF(80)+3FC (40,20),DDQN,LI	***0.907***	***0.923***	0.873	***0.979***	1275.5	1.23

We can also observe how the best performance is obtained when the training paradigm is shifted to a reinforcement learning scheme (DRL+RBFNN) [model: RBF(80)+3FC (40,20),DDQN,CE,LI], with the RBFNN as the policy network and the loss function is replaced by a 0/1 reward scheme based on providing a reward of 1 when the policy network makes the correct prediction and 0 otherwise (Section III.B.2).

The best Recall metrics are achieved with the proposed models while the best alternative machine learning models (SVM+RBF and LM+KA) achieve best Precision metrics. This behavior is also observed with the AWID dataset and is reversed for the UNSW-NB15 dataset. One reason for this behavior is that Normal traffic is the majority class during training for NSL-KDD and AWID, while is the minority class for UNSW-NB15 (Figs. 1-3). The Recall metric is particularly important for NID applications as an important goal is to reduce the number of false negatives.

Fig. 11 provides a graphical representation comparing the F1-score and Accuracy for the best performing models in Table 4. The chart in Fig. 11 shows how the best prediction results are gathered around the proposed RBFNN models with the reinforcement learning variant (DRL+RBFNN) obtaining better results. The next best models are linear models with an expansion of the dimensionality of features using kernel approximation techniques (LM+KA), and SVM models with an RBF kernel. Previous work has already shown that these alternative models are especially suitable for this data set [53].

B. UNSW-NB15 RESULTS

The results for the UNSW-NB15 dataset are given in Table 5. As with the NSL-KDD dataset (Table 4), most of the best results are obtained with a RBFNN model with several subsequent FC layers, with the width of the first dense layer being especially important. The second best result is found to be for the CNN-1D model which seems to be particularly

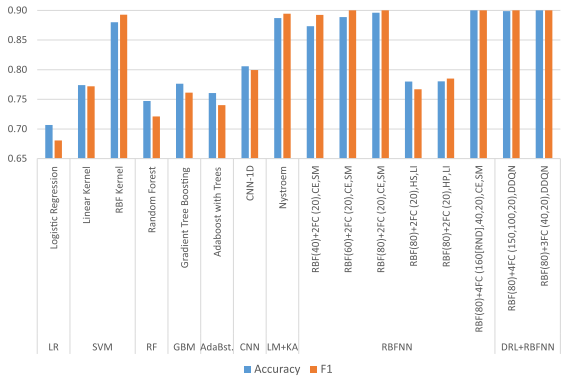

FIGURE 11. Chart showing a comparison of the Accuracy and F1 metrics for the best models applied to the NSL-KDD dataset.

appropriate for this dataset as shown in [52]. The best result is for the extended RBFNN model with additional layers [model: RBF(80)+4FC (160[RND],40,20),CE,SM]. This is a model with untrained random weights in the last layer, similar to an ELM network which has already shown excellent results in classification [55]. It is interesting to appreciate the good behavior of these ELM-like networks, also showing good performance results for the other two datasets. In the original ELM, it is the first layer that has untrained random weights, and the last layer is trained with a linear regression algorithm (without gradient descent), in our case the architecture is different, but maintains the fundamental property of performing a random projection of the data with a shallow (and wide) neural network.

Most RBFNN models perform better than alternative ML models. In this case, the RBFNN models trained without a DRL framework perform better. Similar to the NSD-KDD and AWID datasets, the best results are obtained with a cross-entropy loss.

Fig. 12 provides a chart comparing the Accuracy and F1-score for the best models in Table 5. As with the NSL-KDD dataset, the best results are grouped around the proposed models. The next best results correspond to the CNN and Random Forest models within the alternative ML models. The best model is an RBFNN with several dense trapezoidal-shaped layers (the first layers wider than the last with a monotonic reduction in width size), this behavior is similar to that observed for the NSL-KDD dataset. It is also interesting how the inclusion of an initial non-trainable random layer (similar to an ELM model) can improve the prediction results.

C. AWID RESULTS

The results for the AWID dataset are given in Table 6. These results correspond to a multiclass (4 classes) classification with an extremely unbalanced dataset. We see how the RBFNN models have some difficulties with extremely unbalanced datasets, and how the reinforcement learning training framework helps in this scenario. In this case, the best results correspond to the RBFNN models trained with a reinforcement learning approach (RBFNN+DRL) [model:

TABLE 5. Performance metrics for all models applied to the UNSW-NB15 dataset. Detailed results with column-wise color-coded metrics (from green to red, the greener the better). In addition, the two best values per column are marked in bold-italic.

Model			Accuracy	F1	Precision	Recall	Exec. Times (sec)	
Category		Model detail					Training	Test
Alternative ML models	LR	Logistic Regression	0.843	0.868	0.809	0.938	***8.2***	***0.10***
	SVM	Linear Kernel	0.846	0.870	0.815	0.933	***15.3***	***0.09***
		RBF Kernel	0.815	0.855	0.752	***0.992***	2475.5	644.48
	RF	Random Forest	0.879	0.900	0.828	0.985	81.6	14.34
	GBM	Gradient Tree Boosting	0.857	0.884	0.801	***0.985***	154.6	0.45
	AdaBoost	Adaboost with Trees	0.852	0.880	0.795	0.984	81.1	0.45
	NN	Neural Net (MLP), 3FC (200,50),CE,SM	0.866	0.889	0.814	0.981	261.0	0.97
	CNN	CNN-1D	0.898	***0.913***	0.855	0.980	1393.4	5.48
	LM+KA	Nystroem	0.885	0.895	0.901	0.888	611.5	2.80
Extended RBFNN models	RBFNN	RBF(40)+1FC,CE,SM	0.893	0.903	0.903	0.902	710.1	2.51
		RBF(80)+1FC,CE,SM	0.898	0.905	0.924	0.888	935.6	7.03
		RBF(80)+2FC (20),CE,SM	0.894	0.901	0.928	0.875	1159.0	5.83
		RBF(80)+4FC (160,40,20),CE,SM	***0.901***	0.906	***0.950***	0.866	1488.9	5.34
		RBF(80)+4FC (200,40,20),CE,SM	0.901	0.906	***0.947***	0.869	1513.3	5.43
		RBF(80)+4FC (160[RND],40,20),CE,SM	***0.908***	***0.914***	0.940	0.890	972.5	6.09
	DRL+RBFNN	RBF(80)+4FC (150,100,20),DDQN,LI	0.888	0.899	0.891	0.907	4625.7	7.38
		RBF(80)+3FC (100,20),DDQN,LI	0.852	0.868	0.853	0.884	4838.9	6.58

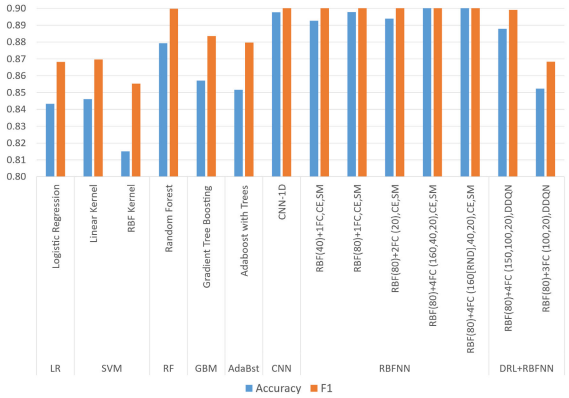

FIGURE 12. Chart showing a comparison of the Accuracy and F1 metrics for the best models applied to the UNSW-NB15 dataset.

TABLE 6. Performance metrics for all models applied to the AWID dataset. Detailed results with column-wise color-coded metrics (from green to red, the greener the better). In addition, the two best values per column are marked in bold-italic.

Model			Accuracy	F1	Precision	Recall	Exec. Times (sec)	
Category		Model detail					Training	Test
Alternative ML models	LR	Logistic Regression	0.951	0.927	***0.953***	0.951	***82.9***	***0.13***
	SVM	Linear Kernel	0.925	0.919	0.917	0.925	***145.1***	***0.15***
	RF	Random Forest	0.945	0.927	***0.946***	0.945	272.7	0.74
	GBM	Gradient Tree Boosting	0.946	0.923	0.914	0.946	521.8	0.70
	AdaBoost	Adaboost with Trees	0.647	0.723	0.836	0.647	229.8	11.25
	NN	Neural Net (MLP), 4FC (100,100,20),CE,SM	0.922	0.886	0.865	0.922	1367.8	6.36
	CNN	CNN-1D	0.913	0.922	0.943	0.913	4068.8	15.81
	LM+KA	Nystroem	0.951	0.927	0.904	0.951	1584.5	3.75
Extended RBFNN models	RBFNN	RBF(80)+2FC (20),CE,SM	0.925	0.894	0.866	0.925	2203.8	11.79
		RBF(5)+2FC (10),CE,SM	0.924	0.920	0.919	0.924	1391.9	8.17
		RBF(10)+2FC (10),CE,SM	0.948	0.926	0.905	0.948	2619.6	8.78
		RBF(10)+4FC (20[RND],20,20),CE,SM	0.916	0.899	0.884	0.916	607.8	9.00
	DRL+RBFNN	RBF(80)+3FC (80,20),DDQN,LI	***0.955***	***0.934***	0.914	***0.955***	4438.4	10.55
		RBF(80)+3FC (100,20),DDQN,LI	***0.954***	***0.938***	0.926	***0.954***	4738.4	12.64

RBF(80)+3FC (100,20),DDQN,LI], followed by Random Forest, Multiclass-Logistic regression (LR) and linear models with a kernel approximation (LM+KA).

Fig. 13 provides a chart comparing the Accuracy and F1-score for the best models in Table 6. The RBFNN+DRL models achieve the best results in Accuracy, F1-score and Recall metrics. The good results obtained when applying reinforcement learning to RBFNN can be justified by the initial exploration phase with an ε-greedy algorithm that avoids easily falling into the local minima forced by the large proportion of Normal traffic.

D. CICIDS2017 RESULTS

The results for the CICIDS2017 dataset are given in Table 7. They correspond to a multiclass (4 classes) classification.

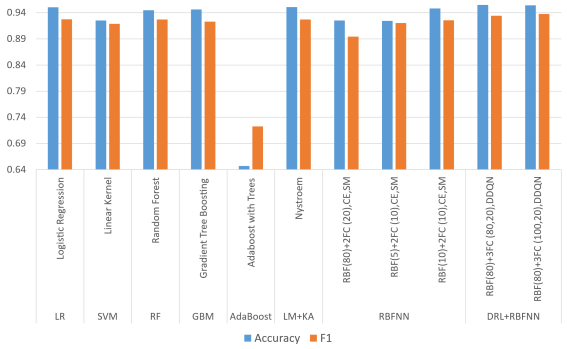

FIGURE 13. Chart showing a comparison of the Accuracy and F1 metrics for the best models applied to the AWID dataset.

TABLE 7. Performance metrics for all models applied to the CICIDS2017 dataset. Detailed results with column-wise color-coded metrics (from green to red, the greener the better). In addition, the two best values per column are marked in bold-italic.

Model			Accuracy	F1	Precision	Recall	Exec. Times (sec)	
Category		Model detail					Training	Test
Alternative ML models	LR	Logistic Regression	0.996	0.749	0.748	0.749	*27.2*	*0.08*
	SVM	Linear Kernel	0.989	0.742	0.745	0.739	*46.3*	*0.05*
	RF	Random Forest	0.998	0.886	0.971	0.848	138.4	0.27
	GBM	Gradient Tree Boosting	***0.998***	0.880	0.990	0.839	326.0	0.14
	AdaBoost	Adaboost with Trees	0.675	0.428	0.635	0.395	124.9	2.95
	NN	Neural Net (MLP), 4FC (100,100,20),CE,SM	***0.998***	0.886	0.995	0.844	640.2	1.40
	CNN	CNN-1D	0.997	***0.997***	***0.997***	***0.997***	1472.8	3.34
	LM+KA	Nystroem	0.993	0.992	0.990	0.993	471.4	0.87
Extended RBFNN models	RBFNN	RBF(80)+2FC (20),CE,SM	0.997	0.997	0.997	0.997	954.8	3.09
		RBF(5)+2FC (10),CE,SM	0.997	***0.997***	0.997	0.997	374.2	2.06
		RBF(10)+2FC (10),CE,SM	0.995	0.994	0.993	0.995	682.6	2.31
		RBF(10)+4FC (20[RND],20,20),CE,SM	0.997	0.997	***0.997***	***0.997***	253.7	2.37
	DRL+RBFNN	RBF(80)+3FC (80,20),DDQN,LI	0.997	0.996	0.996	0.997	1717.1	2.88
		RBF(80)+3FC (100,20),DDQN.LI	0.996	0.994	0.993	0.996	1748.0	3.40

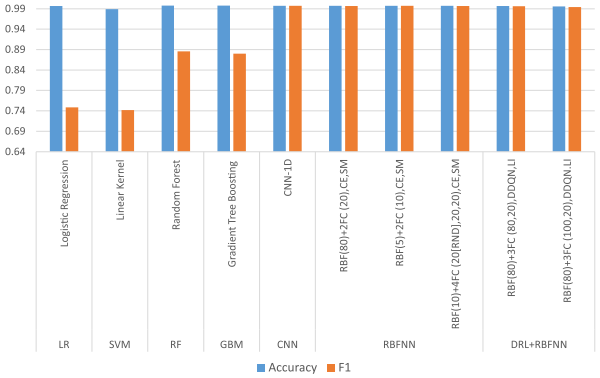

FIGURE 14. Chart showing a comparison of the Accuracy and F1 metrics for the best models applied to the CICIDS2017 dataset.

Fig. 14 provides a chart comparing the Accuracy and F1-score for the best models in Table 7.

The CNN and RBFNN models show the best prediction results (F1-score, Precision and Recall). The DRL + RBFNN models also perform well, but not as good as the simpler RBFNN models.

E. CICDDOS2019 RESULTS

The results for the CICDDOS2019 dataset are given in Table 8. They also correspond to a multiclass (4 classes) classification. Fig. 15 provides a chart comparing the Accuracy and F1-score for the best models in Table 8.

The RBFNN models show the best prediction results (F1-score, Precision and Recall). The CNN and DRL + RBFNN models provide second best results

TABLE 8. Performance metrics for all models applied to the CICDDOS2019 dataset. Detailed results with column-wise color-coded metrics (from green to red, the greener the better). In addition, the two best values per column are marked in bold-italic.

Model			Accuracy	F1	Precision	Recall	Exec. Times (sec)	
Category		Model detail					Training	Test
Alternative ML models	LR	Logistic Regression	0.997	0.845	0.889	0.823	*19.3*	*0.09*
	SVM	Linear Kernel	0.997	0.846	0.894	0.824	*29.9*	*0.04*
	RF	Random Forest	***0.997***	0.830	0.908	0.803	82.5	0.20
	GBM	Gradient Tree Boosting	0.992	0.588	0.752	0.561	273.6	0.19
	AdaBoost	Adaboost with Trees	0.865	0.579	0.745	0.772	81.3	2.60
	NN	Neural Net (MLP), 4FC (100,100,20),CE,SM	***0.997***	0.860	0.896	0.839	550.4	1.32
	CNN	CNN-1D	0.995	0.995	0.995	0.995	1372.5	3.39
	LM+KA	Nystroem	0.992	0.989	0.989	0.992	463.3	0.92
Extended RBFNN models	RBFNN	RBF(80)+2FC (20),CE,SM	0.994	0.994	***0.995***	0.994	621.4	2.94
		RBF(5)+2FC (10),CE,SM	0.997	***0.995***	0.994	***0.997***	529.3	2.13
		RBF(10)+2FC (10),CE,SM	0.991	0.991	0.994	0.991	334.0	2.17
		RBF(10)+4FC (20[RND],20,20),CE,SM	0.997	***0.996***	***0.996***	***0.997***	201.7	2.24
	DRL+RBFNN	RBF(80)+3FC (80,20),DDQN,LI	0.707	0.817	0.988	0.707	1568.6	3.01
		RBF(80)+3FC (100,20),DDQN.LI	0.996	0.994	0.993	0.996	1581.4	3.06

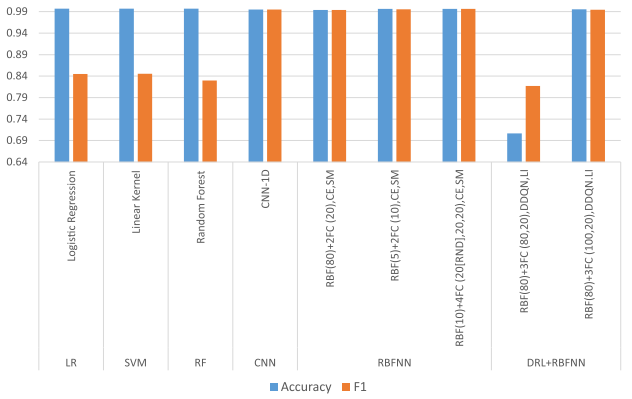

FIGURE 15. Chart showing a comparison of the Accuracy and F1 metrics for the best models applied to the CICDDOS2019 dataset.

F. IMPACT OF NUMBER OF KERNEL FUNCTIONS

It is interesting to investigate the importance of the number of RBF neurons. Fig. 16 provides a chart showing the evolution of the ROC-AUC score for the NSL-KDD and UNSW-NB15 datasets and for different numbers of RBF neurons. We have chosen the ROC-AUC to perform this comparison, as we did not want to further calibrate the results, and the ROC-AUC is a model performance metric that is independent of the probability separation threshold. An interesting conclusion from Fig. 16 is that the number of RBF neurons is much less important than it initially appears, as long as that number is greater than a certain threshold that is dataset dependent. This might seem counterintuitive, since, in its extreme case, having a number of neurons as large as the number of training samples would allow a perfect fit, however, this extreme case would also imply an architecture prone to overfitting with a behavior similar to a K-Nearest Neighbors with a k equal to 1 [56].

G. GENERIC RESULTS

To give a joint vision of the results, in Table 9 are shown the confusion matrices for four datasets. We can observe the difficulties in detecting some of the minority classes for the AWID dataset, which is the most unbalanced of the five datasets, and is the one that benefits the most from joining reinforcement learning training with the RBFNN architecture. Despite this, the proposed models obtain the best classification performance metrics for this dataset. The confusion matrices for NSL-KDD and UNSW-NB15 are quite different,

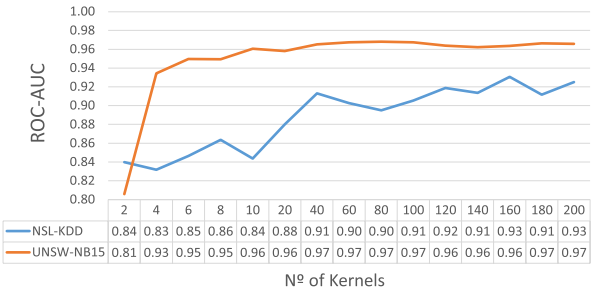

FIGURE 16. Evolution of the ROC-AUC metric vs. the number of RBF neurons (first hidden layer), for two datasets: NSL-DD and UNSW-NB15.

TABLE 9. Confusion matrices for the NSL-KDD, UNSW-NB15, AWID and CICDDoS2019 datasets using the best proposed models, as shown in Sections IV.A-C2.

NSL		Predicted	
		Normal	Attack
Real	Normal	7887	1824
	Attack	274	12559

UNSW-NB15		Predicted	
		Normal	Attack
Real	Normal	34412	2588
	Attack	4993	40339

AWID		Predicted			
		Normal	Impers.	Flood.	Inject.
Real	Normal	527856	2615	313	0
	Impersonation	1018	19061	0	0
	Flooding	5648	84	2365	0
	Injection	16682	0	0	0

CICIDS2017		Predicted			
		Normal	Bot	DDoS	PortScan
Real	Normal	81867	30	54	8
	Bot	251	123	0	0
	DDoS	49	0	25341	0
	PortScan	14	0	9	31924

CICDDoS2019		Predicted			
		Normal	Syn	Normal	UDPLag
Real	Normal	705	44	0	37
	Syn	1	111294	10	17
	UDP	9	3	22312	69
	UDPLag	4	6	238	135

with a large proportion of correct classifications for the majority and minority classes. The confusion matrices for the CICIDS2017 and CICDDoS2019 datasets are also presented. These datasets are also multiclass and unbalanced, although not as unbalanced as AWID. We can see that for both datasets, the vast majority of the predictions are correct for all labels. The best results for CICIDS2017 and CICDDoS2019 are also obtained with an extended RBFNN model (without using DRL), while the best model for AWID is a DRL+RBFNN model.

To ensure that the best results obtained by the proposed models can be considered superior to the alternative machine learning models, Table 10 provides the p-values obtained by applying the Wilcoxon signed-rank test to verify whether the best F1-score obtained with the proposed model is greater than the results obtained with alternative machine learning models. This test is applied separately to each dataset. The p-values obtained confirm that the proposed models offer an F1-score higher than the alternative models with a

TABLE 10. Significance of results using Wilcoxon signed-rank test to verify a higher F1-score of the best proposed model compared to alternative ML models.

Model	p-value	Significance Level (1%)
NSL-KDD	0.20%	Yes
UNSW-NB15	0.20%	Yes
AWID	0.69%	Yes
CICIDS2017	1.11%	No
CICDDoS2019	0.39%	Yes

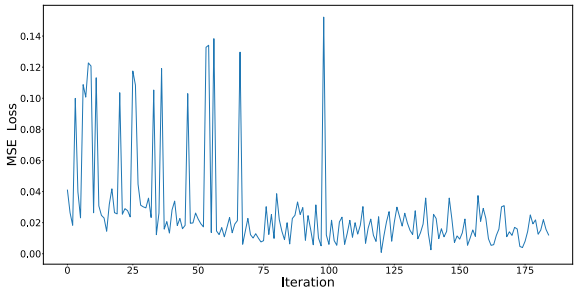

FIGURE 17. Loss (mean-squared error) evolution for Q-learning during training for the AWID dataset (DRL+RBFNN model).

significance level of 1%, for all datasets with the exception of CICIDS2017 where the results are also superior but with a significance level slightly higher (1.11%).

It is interesting to show the convergence of the DRL+RBFNN models during training (Section III.B.2).This convergence is appreciated in Fig. 17, which shows the evolution during training for the loss function of the Q-learning algorithm used to train the RBFNN inside a DRL framework. We see the usual noisy optimization path in these models and how the initial phase is noisier due to the greater effect of the initial exploration phase with an ε-greedy algorithm [7]. Fig. 17 is obtained with [model: RBF(80)+3FC (100,20),DDQN,LI] and the AWID dataset.

It is also important to examine the complexity [57] of the different algorithms and their correspondence with their prediction performance. Fig. 18 provides a graph showing the number of parameters of the different models together with their F1-score. Fig. 18 is done for the AWID dataset, but similar results are obtained for the other datasets. The number of parameters of the models based on neural networks corresponds to their number of weights. For random forest, gradient boosting and AdaBoost, we have used the number of nodes of all the trees built by their respective algorithms as the number of parameters. For linear models (logistic regression and linear SVM) the parameters are directly the parameters used by the models. The number of parameters is not a perfect indicator of complexity, as other factors must be considered [57]: (a) pre-processing of features (e.g., kernel approximation); (b) optimization algorithm (e.g., boosting models); (c) ensemble architecture (e.g., random forest); (d) number of iterations required by the learning process (e.g., number of epochs required by the different NN models); (e) inclusion of an initial exploration/exploitation phase that

FIGURE 18. Comparison of F1-score and complexity (number of parameters) of all models applied to AWID (Section IV.C).

tends to slow down the training phase (e.g., reinforcement learning). However, and despite its limitations, the number of parameters can be used as a first approximation to model complexity [57], and from this point of view, Fig. 18 is appropriate to compare the different NN models and determine that the proposed models provide a good balance between complexity and prediction performance.

Considering all the previous results obtained by applying the proposed RBFNN architectures to the different challenges presented by the five datasets, the main conclusions are:

(a) The extended RBFNN architectures proposed in this work obtain the best performance metrics. The extended DRL+RBFNN achieves the best performance in two of the five scenarios, and the extended RBFNN architecture obtains the best results in the third scenario. The DRL learning scheme applied to RBFNN networks seems to stand out over other architectural alternatives for highly unbalanced datasets (AWID) or noisy label (NSL-KDD) scenarios.

(b) The importance of the network structure after the initial RBF layer, showing that the addition of extra dense layers improves performance. The best architecture for the dense layer block appears to be a trapezoidal shape with a descending number of layers and with the first dense layer being wider than the RBF layer.

(c) The great impact of the loss function and the learning approach (supervised vs. reinforcement). In particular, how using a DRL learning approach can provide better results mainly with extremely unbalanced datasets, and how the hinge-loss in its various configurations does not provide good results.

(d) The limited impact of the number of RBF neurons on performance, as far as this number is greater than a certain threshold which is generally not large.

(e) The best execution times required for training and prediction are obtained with linear ML models (logistic regression and linear SVM). The proposed models are in an intermediate position in terms of execution times, with the DRL + RBFNN models being the worst in this regard. Models based on CNN's architectures offer prediction capabilities similar to the proposed models with generally higher execution times.

We implemented all the neural network models (NN, RBFNN, DRL+RBFNN) in python using Tensorflow/Keras [43], and the rest of ML models in python using the scikit-learn package [54].

H. CHALLENGES AND LIMITATIONS

It is important to explicitly mention the challenges and limitations of this work, which are:

- The difficulties of DRL methods when increasing the dimensionality of the action space (number of labels) are known. This is an issue that should be considered and addressed in future research.

- The additional prediction performance of these methods comes with additional RBF layer complexity and increased processing time, mainly for the training stage.

- A good initial assignment of the RBF cluster centers is important, and introducing additional methods to optimize this initial step is challenging.

- The execution times of the proposed models are high, but lower than the required times of alternative deep learning models (e.g., CNN-1D) with similar prediction performance.

V. CONCLUSION

Network intrusion detection is an increasingly important problem in modern data networking, and it is an active research field in which many types of machine learning and deep learning models have been applied. We propose novel extensions to the RBFNN model. These extensions are based on an end-to-end training scheme using gradient descent for all the parameters of the network: the network weights, and the centers and dispersion parameters of the radial basis functions. This end-to-end training scheme allows us to propose several alternative loss functions, different from the cross-entropy generally used for classification. It also allows a complete RBFNN network to be included as the policy network of an offline reinforcement learning model where the loss function is replaced by a reward function that is not necessarily differentiable. This approach also offers the opportunity to include additional dense hidden layers after the initial RBF layer.

We show through extensive analysis of results, with five different intrusion detection datasets, that the proposed RBFNN extended architectures achieve the best results in most of the performance metrics for the five proposed datasets. We also show that the inclusion of additional dense layers and the application of the reinforcement learning framework for training are both crucial to increasing the performance metrics of the models. The reinforcement learning scheme applied to RBFNN networks stands out above other architectural alternatives, especially in the case of highly unbalanced and/or noisy datasets which are common in network intrusion detection.

As promising directions for future work, we propose: (a) To continue with this line of research, particularly in exploring new loss functions and their application to RBFNN end-to-end training, e.g., contrastive loss. (b) It would also

be of interest to analyze the impact of alternative reinforcement learning algorithms e.g., actor-critic, policy-gradient, adversarial [37]. (c) To apply eXplainable Artificial Intelligence (XAI) techniques to extrapolate the particular policy/classification procedure inferred by the proposed algorithms [58].

REFERENCES

[1] A. Aldweesh, A. Derhab, and A. Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues," *Knowl.-Based Syst.*, vol. 189, Feb. 2020, Art. no. 105124, doi: 10.1016/j.knosys.2019.105124.

[2] A. Drewek-Ossowicka, M. Pietrołaj, and J. Rumiński, "A survey of neural networks usage for intrusion detection systems," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 1, pp. 497–514, Jan. 2021, doi: 10.1007/s12652-020-02014-x.

[3] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102767, doi: 10.1016/j.jnca.2020.102767.

[4] S. M. Tahsien, H. Karimipour, and P. Spachos, "Machine learning based solutions for security of Internet of Things (IoT): A survey," *J. Netw. Comput. Appl.*, vol. 161, Jul. 2020, Art. no. 102630, doi: 10.1016/j.jnca.2020.102630.

[5] D. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, no. 3, pp. 321–355, 1988.

[6] F. Schwenker, H. A. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks," *Neural Netw.*, vol. 14, no. 4, pp. 439–458, 2001, doi: 10.1016/S0893-6080(01)00027-2.

[7] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," *Expert Syst. Appl.*, vol. 141, Mar. 2020, Art. no. 112963, doi: 10.1016/j.eswa.2019.112963.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: Bradford Book, 2018.

[9] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020, *arXiv:2005.01643*.

[10] A. AbuGhazleh, M. Almiani, B. Magableh, and A. Razaque, "Intelligent intrusion detection using radial basis function neural network," in *Proc. 6th Int. Conf. Softw. Defined Syst. (SDS)*, Jun. 2019, pp. 200–208, doi: 10.1109/SDS.2019.8768575.

[11] R. Ma, Y. Liu, X. Lin, and Z. Wang, "Network anomaly detection using RBF neural network with hybrid QPSO," in *Proc. IEEE Int. Conf. Netw., Sens. Control*, Apr. 2008, pp. 1284–1287, doi: 10.1109/ICNSC.2008.4525415.

[12] H. Sheth, P. B. Shah, and S. Yagnik, "A survey on RBF neural network for intrusion detection system," *Int. J. Eng. Res. Appl.*, vol. 4, no. 12, pp. 17–22, 2014.

[13] U. Ahmed and A. Masood, "Host based intrusion detection using RBF neural networks," in *Proc. Int. Conf. Emerg. Technol.*, Oct. 2009, pp. 48–51, doi: 10.1109/ICET.2009.5353204.

[14] J. Bi, K. Zhang, and X. Cheng, "Intrusion detection based on RBF neural network," in *Proc. Int. Symp. Inf. Eng. Electron. Commerce*, May 2009, pp. 357–360, doi: 10.1109/IEEC.2009.80.

[15] J. Chenou, G. Hsieh, and T. Fields, "Radial basis function network: Its robustness and ability to mitigate adversarial examples," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2019, pp. 102–106, doi: 10.1109/CSCI49370.2019.00024.

[16] P. Yichun, N. Yi, and H. Qiwei, "Research on intrusion detection system based on IRBF," in *Proc. 8th Int. Conf. Comput. Intell. Secur.*, Nov. 2012, pp. 544–548, doi: 10.1109/CIS.2012.128.

[17] Y. Hu, J. J. You, J. N. K. Liu, and T. He, "An eigenvector based center selection for fast training scheme of RBFNN," *Inf. Sci.*, vol. 428, pp. 62–75, Feb. 2018, doi: 10.1016/j.ins.2017.08.092.

[18] Z. Yang, X. Wei, L. Bi, D. Shi, and H. Li, "An intrusion detection system based on RBF neural network," in *Proc. 9th Int. Conf. Comput. Supported Cooperat. Work Design*, vol. 2, 2005, pp. 873–875, doi: 10.1109/CSCWD.2005.194301.

[19] Z. Chen and P. Qian, "Application of PSO-RBF neural network in network intrusion detection," in *Proc. 3rd Int. Symp. Intell. Inf. Technol. Appl.*, vol. 1, 2009, pp. 362–364, doi: 10.1109/IITA.2009.154.

[20] S. Mohammadi and F. Amiri, "An efficient hybrid self-learning intrusion detection system based on neural networks," *Int. J. Comput. Intell. Appl.*, vol. 18, no. 1, Mar. 2019, Art. no. 1950001, doi: 10.1142/S1469026819500019.

[21] L. Lv, W. Wang, Z. Zhang, and X. Liu, "A novel intrusion detection system based on an optimal hybrid kernel extreme learning machine," *Knowl.-Based Syst.*, vol. 195, May 2020, Art. no. 105648, doi: 10.1016/j.knosys.2020.105648.

[22] X. Tong, Z. Wang, and H. Yu, "A research using hybrid RBF/Elman neural networks for intrusion detection system secure model," *Comput. Phys. Commun.*, vol. 180, no. 10, pp. 1795–1801, 2009, doi: 10.1016/j.cpc.2009.05.004.

[23] L.-Y. Tian and W.-P. Liu, "Incremental intrusion detecting method based on SOM/RBF," in *Proc. Int. Conf. Mach. Learn. Cybern.*, Jul. 2010, pp. 2849–2853, doi: 10.1109/ICMLC.2010.5580770.

[24] Y. Kadhim and A. Mishra, "Radial basis function (RBF) based on multistage autoencoders for intrusion detection system (IDS)," in *Proc. 1st Int. Informat. Softw. Eng. Conf. (UBMYK)*, Nov. 2019, pp. 1–4, doi: 10.1109/UBMYK48245.2019.8965627.

[25] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, no. 9, pp. 1481–1497, Sep. 1990, doi: 10.1109/5.58326.

[26] Y.-M. Du, B.-B. Yan, Y.-C. Jiang, and Q.-S. Liu, "Reinforcement learning method based on radial basis function neural network," in *Design, Manufacturing and Mechatronics*. Singapore: World Scientific, 2016, pp. 21–27.

[27] R. S. Williem and K. Setiawan, "Reinforcement learning combined with radial basis function neural network to solve job-shop scheduling problem," in *Proc. IEEE Int. Summer Conf. Asia Pacific Bus. Innov. Technol. Manage.*, Jul. 2011, pp. 29–32, doi: 10.1109/APBITM.2011.5996285.

[28] S. Slušný, R. Neruda, and P. Vidnerová, "Comparison of RBF network learning and reinforcement learning on the maze exploration problem," in *Artificial Neural Networks—ICANN 2008*. Berlin, Germany: Springer, 2008, pp. 720–729.

[29] K. Asadi, N. Parikh, R. E. Parr, G. D. Konidaris, and M. L. Littman, "Deep radial-basis value functions for continuous control," 2020, *arXiv:2002.01883*.

[30] N. Shafiabady, M. A. N. Vakilian, and D. Isa, "An adaptive controller using radial basis function neural network with reinforcement learning," *Commun. Appl. Electron.*, vol. 1, no. 7, pp. 7–13, May 2015. Accessed: Mar. 14, 2021. [Online]. Available: http://www.caeaccess.org

[31] J. Li, J. Yi, D. Zhao, and G. Xi, "A reinforcement learning based radial-bassis function network control system," in *Advances in Neural Networks—ISNN 2005*. Berlin, Germany: Springer, 2005, pp. 640–645.

[32] C.-G. Li, M. Wang, Z.-J. Huang, and Z.-F. Zhang, "An actor-critic reinforcement learning algorithm based on adaptive RBF network," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 2, Jul. 2009, pp. 984–988, doi: 10.1109/ICMLC.2009.5212431.

[33] Y.-H. Cheng, J.-Q. Yi, and D.-B. Zhao, "Application of actor-critic learning to adaptive state space construction," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 5, 2004, pp. 2985–2990, doi: 10.1109/ICMLC.2004.1378544.

[34] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018, *arXiv:1802.09089*.

[35] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "A hierarchical hybrid intrusion detection approach in iot scenarios," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–7, doi: 10.1109/GLOBECOM42002.2020.9348167.

[36] T. Thi Nguyen and V. Janapa Reddi, "Deep reinforcement learning for cyber security," 2019, *arXiv:1906.05799*.

[37] G. Caminero, M. Lopez-Martin, and B. Carro, "Adversarial environment reinforcement learning algorithm for intrusion detection," *Comput. Netw.*, vol. 159, pp. 96–109, Aug. 2019, doi: 10.1016/j.comnet.2019.05.013.

[38] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 53–58.

[39] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6, doi: 10.1109/MilCIS.2015.7348942.

[40] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 184–208, 1st. Quart., 2016, doi: 10.1109/COMST.2015.2402161.
[41] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116, doi: 10.5220/0006639801080116.
[42] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2019, pp. 1–8, doi: 10.1109/CCST.2019.8888419.
[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and S. Ghemawat, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.
[44] F. Schwenker, H. A. Kestler, G. Palm, and M. Hoher, "Similarities of LVQ and RBF learning—A survey of learning rules and the application to the classification of signals from high-resolution electrocardiography," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 1, Oct. 1994, pp. 646–651, doi: 10.1109/ICSMC.1994.399913.
[45] K. Z. Mao, "RBF neural network center selection based on Fisher ratio class separability measure," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1211–1217, Sep. 2002, doi: 10.1109/TNN.2002.1031953.
[46] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.
[47] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," 2015, *arXiv:1509.06461*.
[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[50] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Serv. Appl.*, vol. 9, no. 1, p. 16, Jun. 2018, doi: 10.1186/s13174-018-0087-2.
[51] K. A. P. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, "Internet of Things: A survey on machine learning-based intrusion detection approaches," *Comput. Netw.*, vol. 151, pp. 147–157, Mar. 2019, doi: 10.1016/j.comnet.2019.01.023.
[52] M. Azizjon, A. Jumabek, and W. Kim, "1D CNN based network intrusion detection with normalization on imbalanced data," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Feb. 2020, pp. 218–224, doi: 10.1109/ICAIIC48513.2020.9064976.
[53] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Shallow neural network with kernel approximation for prediction problems in highly demanding data networks," *Expert Syst. Appl.*, vol. 124, pp. 196–208, Jun. 2019, doi: 10.1016/j.eswa.2019.01.063.
[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011, Accessed: Sep. 17, 2020. [Online]. Available: http://scikit-learn.sourceforge.net
[55] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006, doi: 10.1016/j.neucom.2005.12.126.
[56] J. E. B. Maia, V. R. S. Laboreiro, F. E. Chaves, F. J. A. Maia, T. G. N. Silva, and T. N. Ferreira, "Performance comparison between edited kNN and MQ-RBFN for regression and classification tasks," in *Proc. 11st Brazilian Congr. Comput. Intell.*, Mar. 2016, pp. 1–15, doi: 10.21528/CBIC2013-319.
[57] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, "Model complexity of deep learning: A survey," *Knowl. Inf. Syst.*, vol. 63, no. 10, pp. 2585–2619, Oct. 2021, doi: 10.1007/s10115-021-01605-0.
[58] A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescape, "XAI meets mobile traffic classification: Understanding and improving multimodal deep learning architectures," *IEEE Trans. Netw. Service Manage.*, early access, Jul. 19, 2021, doi: 10.1109/TNSM.2021.3098157.

MANUEL LOPEZ-MARTIN (Senior Member, IEEE) received the Ph.D. degree in machine learning applied to network traffic analysis and prediction from the University of Valladolid, Spain, in 2019. He is currently a Research Associate with the University of Valladolid. He has worked as a Data Scientist with Telefonica and has more than 25 years of experience in the development of IT software projects. His research activities involve applying machine learning to intrusion detection in data networks and time-series forecasting.

ANTONIO SANCHEZ-ESGUEVILLAS (Senior Member, IEEE) received the Ph.D. degree in QoS over IP networks from the University of Valladolid, Spain, in 2004. He has managed innovation at Telefonica and has been an Adjunct Professor and an Honorary Collaborator with the University of Valladolid, supervising several Ph.D. students. He has coordinated very large international research and development projects and has over 50 international publications and several patents. His current research interests include digital services and machine learning.

JUAN IGNACIO ARRIBAS received the M.Sc. and Ph.D. degrees in electrical engineering from the University of Valladolid, Spain. He is currently a part of the Department of Electrical Engineering, University of Valladolid, as an Associate Professor. He was an FPI Postgraduate Research Fellow with the Department of Science, Spain. He has visited the University of Maryland, Baltimore, MD, USA, and the Barrow Neurological Institute, Phoenix, AZ, USA. His research interests include machine learning, neural networks, and expert systems and their applications to computer aided diagnosis and computer vision.

BELEN CARRO received the Ph.D. degree in broadband access networks from the University of Valladolid, Spain, in 2001. She is currently a Professor and the Director of the Communications Systems and Networks (SRC) Laboratory, Universidad de Valladolid, working as a Research Manager in NGN communications and services, VoIP/QoS, and machine learning. She has supervised 12 Ph.D. students and has extensive research publications experience as an author, a reviewer, and an editor.