Pattern Recognition 143 (2023) 109805

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Fast deep autoencoder for federated learning

David Novoa-Paradela*, Oscar Fontenla-Romero, Bertha Guijarro-Berdiñas

Universidade da Coruña, CITIC, Campus de Elviña s/n, A Coruña, 15008, Spain

ARTICLE INFO

Article history:
Received 12 July 2022
Revised 4 May 2023
Accepted 5 July 2023
Available online 8 July 2023

Keywords:
Deep autoencoder
Anomaly detection
Federated learning
Edge computing
Machine learning

ABSTRACT

This paper presents a novel, fast and privacy preserving implementation of deep autoencoders. DAEF (Deep AutoEncoder for Federated learning), unlike traditional neural networks, trains a deep autoencoder network in a non-iterative way, which drastically reduces training time. Training can be performed incrementally, in parallel and distributed and, thanks to its mathematical formulation, the information to be exchanged does not endanger the privacy of the training data. The method has been evaluated and compared with other state-of-the-art autoencoders, showing interesting results in terms of accuracy, speed and use of available resources. This makes DAEF a valid method for edge computing and federated learning, in addition to other classic machine learning scenarios.

1. Introduction

As happened at the time with the massive adoption of personal computers, the technological development of recent years has caused a substantial increase in the number of small computing machines such as smartphones or Internet of Things (IoT) devices, for both industrial and personal use. Despite their size, they have enough computing power to perform tasks that a few years ago were considered unapproachable, such as the training of small machine learning models, real-time inference or the exchange of large amounts of information at high speeds. Due to the abundance of these devices and the inefficiencies of traditional cloud computing for applications that demand low latencies, a new computing paradigm called edge computing has emerged [1]. Edge computing (EC) moves computing away from data centers to the edge of the network, bringing cloud computing services and utilities closer to the end user and their devices. This allows faster information processing and response time, as well as freeing up the network bandwidth.

From a machine learning (ML) point of view, this new technological scenario is very suitable for the use of federated learning [2]. Federated learning (FL) is a collaborative machine learning scheme that allows heterogeneous devices with different private datasets to work together to train a global model. In addition, this work scheme emphasizes the preservation of the privacy of local data collected on each device by implementing mechanisms that prevent possible direct and indirect leaks of their data.

On the other hand, in machine learning, anomaly detection (AD) is the branch that builds models capable of differentiating between normal and anomalous data [3]. A priori, this turns anomaly detection into a classification problem with only two classes. However, since anomalies tend to occur sporadically, normal data are the ones that prevail in these scenarios, so, commonly, that models must be trained with only normal data. The objective is to learn to represent the normal class with high precision to be able to classify new data as either normal or abnormal.

Due to its high economic cost, in many real scenarios powerful data centers are not available, and it is necessary to resort to cloud computing services. If the machine learning models are hosted in the cloud, the occurrence of high latencies can negatively affect their purpose. In many real systems, the response time to a detection of an anomaly can be critical, as is the case of failures in industrial systems [4] or network intrusions detection [5]. In addition, in certain scenarios such as the medical or banking field, the privacy of the data that is exchanged is essential. The development of anomaly detection techniques based on edge computing and federated learning may be the solution to reduce these response times and infrastructure limitations while preserving data privacy.

In this paper, we introduce DAEF (Deep AutoEncoder for Federated learning), a fast and privacy-preserving deep autoencoder very suitable for edge computing and federated learning scenarios, in addition to classic machine learning environments. Unlike traditional deep neural networks, its learning method is non-iterative, which drastically reduces its training time. Its training can be car-

* Corresponding author.
E-mail addresses: david.novoa@udc.es (D. Novoa-Paradela), oscar.fontenla@udc.es (O. Fontenla-Romero), berta.guijarro@udc.es (B. Guijarro-Berdiñas).

https://doi.org/10.1016/j.patcog.2023.109805

ried out incrementally (aggregation of models), in a distributed way (training shared among multiple nodes), and in parallel (at the node level if it has several cores), and due to its mathematical formulation the information that is exchanged does not endanger the privacy of the training data. All of this makes DAEF a useful method for edge computing and federated training, capable of performing tasks such as anomaly detection on large datasets while maintaining the performance of traditional (iterative) autoencoders.

This document is structured as follows. Section 2 contains a brief review of the main anomaly detection techniques for edge computing, providing an overview of this field. Section 3 describes the ideas taken as the basis for the development of the proposed DAEF method and Section 4 describes its operation. Section 5 discusses DAEF's privacy-preserving capabilities. Section 6 illustrates the performance of DAEF through a comparative study with traditional autoencoders. Finally, conclusions are drawn in Section 7.

2. Related work

Anomaly detection is a field that has a large number of algorithms that solve the problem of distinguishing between normal and anomalous instances in a wide variety of ways [1,6]. Depending on the assumptions and processes they employ, in traditional anomaly detection, we can distinguish between five main types of methods: probabilistic, distance-based, information theory-based, boundary-based, and reconstruction-based methods. In general, these algorithms are characterized by their high performance when classifying new data, however, they do not focus on other aspects which from a centralized perspective may seem less important, such as data privacy and incremental learning. This makes it difficult to apply many of these classical methods in decentralized environments. For this reason, the strong expansion of edge computing has brought with it a new line of research in the field of anomaly detection in charge of designing new algorithms capable of learning in a distributed and, in some cases, incremental way, while preserving data privacy.

Due to their good performance, it is common for these anomaly detection methods to be based on reconstruction (neural networks). In this section, we will distinguish between reconstruction based methods that use autoencoders [7] and those that do not. Among those that do not use autoencoders is DïOT [8], a self-learning distributed system for the security monitoring of IoT devices which utilizes a novel anomaly detection approach based on representing network packets as symbols, allowing to use a language analysis technique to detect anomalies. Hussain et al.[9]. presented a deep learning framework to monitor user activities of multiple cells and thus detect anomalies using feedforward deep neural networks. Abdel et al.[10]. introduced a federated stacked long short-time memory model to solve multi-task problems using IoT sensors in smart buildings. Zhao et al. [11] propose a multi-task deep neural network in federated learning to perform simultaneously network anomaly detection, VPN traffic recognition, and traffic classification. Other authors like Preuveneers et al. [12] propose the use of blockchain technology to carry out a decentralized registry of federated model updates. This guarantees the integrity of incrementally-learned machine learning models by cryptographically chaining one machine learning model to the next. These solutions obtain good results, however, they do not emphasize privacy preservation and their iterative learning can lead to long training times.

On the other hand, if we focus on autoencoders [7], it is also possible to find works oriented toward edge computing and/or federated learning scenarios. Autoencoders (AE) are a type of self-associative neural network whose output layer seeks to reproduce the data presented to the input layer after having gone through a dimensional compression phase. In this way, they manage to obtain a representation of the input data in a space with a dimension smaller than the original, learning a compact representation of the data, retaining the important information, and compressing the redundant one. For this reason, they are widely used for the elaboration of models that are robust to noise, an important quality in anomaly detection and regression problems. Fig. 1 represents the traditional architecture of an autoencoder.

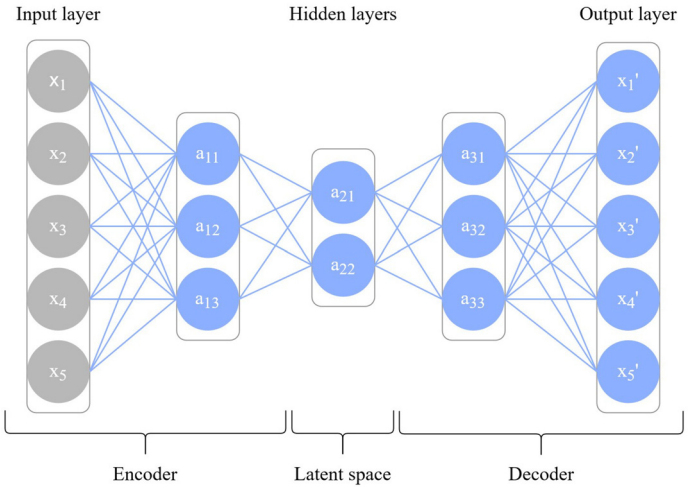

Fig. 1. Example of autoencoder neural network architecture.

Luo and Nagarajan [13] propose to use autoencoders for anomaly detection in wireless sensor networks, however, each edge device does not train a local model with its own data. These devices send their local data to a central cloud node from which the training of the global model is carried out. In the approach presented by Ngo et al. [14], an adaptive hierarchical edge computing system composed of three autoencoder models of increasing complexity is used for IoT anomaly detection.

In the two previous works, as well as in the majority that uses this type of networks, the autoencoders are trained during several iterations to adjust their parameters (weights, bias) using techniques such as the gradient descent and backpropagation. This greatly increases training time, especially when dealing with large datasets or complex networks architectures, which in edge computing scenarios can be critical. However, there is a line of work that allows training autoencoders in a non-iterative way. This is based on Extreme Learning Machines (ELM) [15], an alternative learning algorithm originally formulated for single-hidden layer feedforward neural networks (SLFNs). This algorithm tends to provide good generalization performance and an extremely fast learning speed. Over time, more advanced versions such as MLELM [16], a multilayer version of ELM, or DELM [17], a deep version of ELM, have been developed. For anomaly detection in edge computing and federated learning scenarios, Ito et al. [18] propose combining OS-ELM (Online Sequential Extreme Learning Machine) [19] with autoencoders. This allows each edge device to train its own local model and incrementally update it with the results obtained by the other devices. Nevertheless, a possible limitation of this solution is its autoencoder architecture with only one hidden layer, which in some cases may not be sufficient.

In this work we present DAEF, a deep autoencoder with the following characteristics:

- The architecture is deep (more than one hidden layer) and asymmetrical.
- The training process is non-iterative and therefore faster than a traditional autoencoder.

- It can be trained in a distributed and incremental way, which makes it suitable for edge computing and federated learning environments.
- It is a privacy-preserving method.

3. Background

This section introduces the theoretical foundations and as well as ideas taken as the basis for the development of DAEF.

3.1. Federated learning architectures

Federated Learning systems can be classified into four types based on the level of centralization of their architectures [20]: centralized, hierarchical, regional and decentralized. In centralized FL systems, all edge nodes are connected to a central aggregation node to update local weights and distribute models. Hierarchical architectures achieve less level of centralization by keeping a central node but include several regional aggregation nodes whose goal is to reduce data exchange and manage local devices. Regional architectures eliminate the central node to eliminate the risk of a single point of failure. Decentralized approaches move all tasks to the edge so that edge nodes will handle both local training and model aggregation.

The most common is to use centralized or semi-centralized architectures, in which the aggregator node is a server with high computational capabilities. In this work, we want to design a valid method to be used with more decentralized architectures on FL and EC scenarios, in which the training and aggregations are carried out in the edge devices without requiring large computational power, and exchange of data through the network is secure and preserves the privacy of local datasets.

3.2. Distributed singular value decomposition autoencoder

DSVD-autoencoder (Distributed Singular Value Decomposition Autoencoder) [21] is a hidden single-layer autoencoder network for anomaly detection, consisting of an encoder followed by a decoder. The idea of its encoder will serve as the basis for our work and, therefore, is the one that will be explained in detail below.

Regarding its encoder part, the aim is to learn a vector space embedding of the input data $\mathbf{X} \in \mathbb{R}^{m_0 \times n}$, where m_0 is the number of input variables and n the number of data samples, extracting a meaningful but lower dimensional representation, known as the latent space. This can be accomplished by a low-rank matrix approximation of $\mathbf{X}$, which is a minimization problem that tries to approximate a given matrix by another one subject to the constraint that the approximating matrix has reduced rank [22]. The size of this rank is determined by the size of the hidden layer. Thus, being m_1 the number of neurons of this layer, the Singular Value Decomposition (SVD) of rank-m_1 of matrix $\mathbf{X}$ is used to obtain the weights $\mathbf{W}_1$ of this first layer, as follows. Let's consider the full SVD of $\mathbf{X}$, a factorization of the form:

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{1}$$

where $\mathbf{S} \in \mathbb{R}^{m_0 \times n}$ is a diagonal matrix with descending ordered non-negative values on the diagonal that are the singular values of $\mathbf{X}$, while $\mathbf{U} \in \mathbb{R}^{m_0 \times m_0}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices containing the left and right singular vectors of $\mathbf{X}$. In a low-rank approximation, the optimal rank-m_1 approximation of $\mathbf{X}$ can be computed by taking the first m_1 columns of $\mathbf{U}$ and rows of $\mathbf{V}^T$ and truncating $\mathbf{S}$ to the first m_1 diagonal elements. The new truncated matrices $\mathbf{U}_{m_1} \in \mathbb{R}^{m_0 \times m_1}$ and $\mathbf{V}^T_{m_1} \in \mathbb{R}^{m_1 \times n}$ are, respectively, m_1-dimensional representations of rows (features) and columns (samples) of the input data $\mathbf{X}$. Therefore, $\mathbf{U}_{m_1}$ is used as the weights $\mathbf{W}_1$ for the first layer as it contains the m_1-dimensional transformation of the input space ($\mathbb{R}^{m_0} \rightarrow \mathbb{R}^{m_1}$).

Moreover, in a distributed scenario, the SVD of an entire data matrix $\mathbf{X}$ distributed into P several blocks, that is $\mathbf{X} = \left[\mathbf{X}^1 | \mathbf{X}^2 | \cdots | \mathbf{X}^P\right]$, can be also computed distributively by calculating at each site $p = 1, \ldots, P$ the local matrices $\mathbf{U}^p$ and $\mathbf{S}^p$, corresponding to the rank-m_1 SVD of $\mathbf{X}^p$, and then arbitrarily computing the following operation:

$$[\mathbf{U}_{m_1}, \mathbf{S}_{m_1}, \sim] = SVD([\mathbf{U}^1\mathbf{S}^1 | \ldots | \mathbf{U}^P\mathbf{S}^P]). \tag{2}$$

Notice that, as only the local SVD decompositions are exchanged between nodes, from which the original training data could not be deduced, data privacy is preserved in this process. Algorithm 1 shows this process.

Algorithm 1 Distributed singular value decomposition (DSVD).

Input: $\mathbf{X} = \left[\mathbf{X}^1 | \mathbf{X}^2 | \cdots | \mathbf{X}^P\right] \in \mathbb{R}^{m_0 \times n}$, training data split into P partitions (m_0 variables $\times$ n samples); m_1, rank of the SVD;

Output: $\mathbf{U}_{m_1}$, $\mathbf{S}_{m_1}$ corresponding to the rank-m_1 decomposition of $\mathbf{X}$;

function DSVD
  parallelFor $p = 1 \ldots P$ ▷ Each partition in parallel
    $\left(\mathbf{U}^p, \mathbf{S}^p, \sim\right) = SVD(\mathbf{X}^p)$
  end parallelFor
  $[\mathbf{U}, \mathbf{S}, \sim] = SVD\left(\left[\left(\mathbf{U}^1\mathbf{S}^1 \mid \cdots \mid \mathbf{U}^P\mathbf{S}^P\right)\right]\right)$ ▷ Recalculate
  $\mathbf{U}_{m_1} = \mathbf{U}[:, 1 : m_1]$ ▷ Obtain the rank-m_1
  $\mathbf{S}_{m_1} = \mathbf{S}[:, 1 : m_1]$ ▷ Obtain the rank-m_1
end function

3.3. Regularized One-Layer Neural Network

ROLANN (Regularized One-Layer Neural Networks) [23] is a L2 norm-regularized training method that allows training single-layer neural networks (without hidden layers) in a non-iterative manner by minimizing the mean squared error (MSE) measured before the activation function of the output neurons, as described in [24]. In addition to not being iterative, the complexity of the method depends on the smaller dimensions that define the size of the training set (number of samples and number of variables), which together make the method computationally very efficient. Moreover, the algorithm can be used in an incremental and distributed way while preserving privacy, two possibilities that can be combined as needed to obtain, for instance, a distributed learning environment that also learns incrementally at each location. These characteristics make the method a perfect fit for federated learning environments.

Its bases are as follows. Let's consider the training input data $\mathbf{X} \in \mathbb{R}^{m \times n}$, where m is the number of input variables and n the number of data samples. After propagating this data through the network we can obtain, for each data point, the values of the derivative $\mathbf{f}'$ and the inverse $\bar{\mathbf{d}}$ of the neural function $\mathbf{f}$. With this information, the weights $\mathbf{W}$ of the network can be analytically calculated as:

$$[\mathbf{U}, \mathbf{S}, \sim] = SVD(\mathbf{X}\mathbf{F}), \tag{3}$$

$$\mathbf{M} = \mathbf{X} * \left(\mathbf{f}' .* \mathbf{f}' .* \bar{\mathbf{d}}\right), \tag{4}$$

$$\mathbf{W} = \mathbf{U} * inv\left(\mathbf{S} * \mathbf{S} + \lambda\mathbf{I}\right) * \left(\mathbf{U}^T * \mathbf{M}\right) \tag{5}$$

where $\mathbf{F}$ is the diagonal matrix of $\mathbf{f}$, λ is a weight-decay based regularization parameter and $\mathbf{I}$ is the identity matrix.

As mentioned, this method allows incremental and distributed learning. Suppose $\mathbf{M}^p$, $\mathbf{U}^p$ and $\mathbf{S}^p$ correspond to the knowledge obtained in the current data partition p. If there were a previous k data partition that ROLANN was already trained on, this method can learn from both partitions by calculating:

$$\left[\mathbf{U}^{k|p}, \mathbf{S}^{k|p}, \sim\right] = SVD\left(\mathbf{U}^k\mathbf{S}^k \mid \mathbf{U}^p\mathbf{S}^p\right), \tag{6}$$

$$\mathbf{M}^{k|p} = \mathbf{M}^k + \mathbf{M}^p, \tag{7}$$

and finally obtaining the weights as:

$$\mathbf{W} = \mathbf{U}^{k|p} * inv\left(\mathbf{S}^{k|p} * \mathbf{S}^{k|p} + \lambda\mathbf{I}\right) * \left(\mathbf{U}^{k|p^T} * \mathbf{M}^{k|p}\right), \tag{8}$$

Algorithm 2 shows this process in detail.

Algorithm 2 ROLANN for incremental/distributed regularized learning.

Input: $\mathbf{X}_p \in \mathbb{R}^{m\times n}$, new training data ($m$ variables $\times$ n samples); $\mathbf{d}_p \in \mathbb{R}^{n\times 1}$, desired outputs; f, nonlinear activation function (invertible); λ, regularization hyperparameter; $\mathbf{M}_k, \mathbf{U}_k, \mathbf{S}_k$, matrices calculated using previous data (optional);

Output: $\mathbf{w} \in \mathbb{R}^{m\times 1}$, optimal weights; $\mathbf{M}_{k|p}, \mathbf{U}_{k|p}, \mathbf{S}_{k|p}$, updated $\mathbf{M}, \mathbf{U}, \mathbf{S}$ matrices;

function ROLANN
  $\mathbf{X}_p = \left[ones(1, n); \mathbf{X}_p\right]$ ▷ Bias is added
  $\bar{\mathbf{d}}_p = f^{-1}\left(\mathbf{d}_p\right)$ ▷ Inverse of the neural function
  $\mathbf{f}^{'} = f^{'}\left(\bar{\mathbf{d}}_p\right)$ ▷ Derivate of the neural function
  $\mathbf{F}_p = diag(\mathbf{f}^{'})$ ▷ Diagonal matrix
  if ($\mathbf{M}_k$ & $\mathbf{U}_k$ & $\mathbf{S}_k$ are empty matrices) **then**
    $\left[\mathbf{U}_{k|p}, \mathbf{S}_{k|p}, \sim\right] = SVD(\mathbf{X}_p * \mathbf{F}_p)$ ▷ Economy size
    $\mathbf{M}_{k|p} = \mathbf{X}_p * (\mathbf{f}^{'} .* \mathbf{f}^{'} .* \bar{\mathbf{d}}_p)$
  else ▷ Combine previous knowledge
    $\mathbf{M}_{k|p} = \mathbf{M}_k + \mathbf{X}_p * (\mathbf{f}^{'} .* \mathbf{f}^{'} .* \bar{\mathbf{d}}_p)$
    $\left[\mathbf{U}_p, \mathbf{S}_p, \sim\right] = SVD(\mathbf{X}_p * \mathbf{F}_p)$
    $\left[\mathbf{U}_{k|p}, \mathbf{S}_{k|p}, \sim\right] = SVD([\mathbf{U}_k\mathbf{S}_k | \mathbf{U}_p\mathbf{S}_p])$
  end if
  $\mathbf{w} = \mathbf{U}_{k|p} * inv(\mathbf{S}_{k|p} * \mathbf{S}_{k|p} + \lambda\mathbf{I}) * (\mathbf{U}_{k|p}^T * \mathbf{M}_{k|p})$
end function

3.4. Multilayer Extreme Learning Machine

MLELM (Multilayer Extreme Learning Machine) [16] is a multilayer neural network that, in each layer, makes use of unsupervised learning to train its parameters, eliminating the need to fine-tuning the network. As a result, the authors obtain a method to train multilayer networks in a non-iterative, fast, and mathematically simple way. Specifically, it obtains the weights at each layer by using an ELM-AE (Extreme Learning Machine-Autoencoder) [16], which is an unsupervised single hidden layer autoencoder that, like any other, tries to reproduce the input signal at the output. This mechanism has served as an inspiration for the work presented here.

4. The proposed method

The main objective DAEF (Deep Autoencoder for Federated learning) is to learn a compressed representation of the normal data and, from this reduced space, to reconstruct the inputs at the output of the autoencoder. These tasks should be carried out in a distributed way, and incrementally where possible, to apply the algorithm in edge computing and federated learning environments. To achieve this, DAEF employs an asymmetric autoencoder architecture as shown in Fig. 2. A first single-layer encoder reduces the dimensionality of the input data and it is adjusted using a distributed SVD process. It is followed by a multi-layer decoder to reconstruct the input signal at the output which is trained in layer-by-layer basis through a non-iterative process. This section presents in detail the steps followed by the method and its theoretical foundations.

4.1. The encoder

Given an input dataset $\mathbf{X} \in \mathbb{R}^{m_0\times n}$, where m_0 is the number of input variables and n the number of data samples, the goal of the encoder is to transform $\mathbf{X}$ into a vector space embedding of lower dimension. This will be done using the Distributed Singular Value Decomposition (Algorithm 1). Therefore, the weights of the first layer $\mathbf{W}_1 = \mathbf{U}_1$ are obtained collaboratively across all node locations into which data is partitioned. Finally, the outputs of the first hidden layer of the network can be calculated, at each location p, as:

$$\mathbf{H}_1^p = f_1\left(\mathbf{W}_1^T\mathbf{X}^p\right); \forall p = 1, \ldots, P, \tag{9}$$

where f_1 is the activation function of the first hidden layer.

It has been decided to use a single-layer encoder, i.e., a single dimensionality reduction using SVD, as chaining several SVD processes sequentially and progressively (one per layer) did not show better performance. Since the encoder consists of a single layer and the decoder of several layers, the architecture of this autoencoder is asymmetrical, as can be seen in Fig. 2.

4.2. The decoder

In the decoder, the goal is to reconstruct the input from the low-dimensional representation provided by the output of the first hidden layer $\mathbf{H}_1$ (see Eq. 9). In order to be able to work with large datasets in a fast and efficient way, we propose to apply a non-iterative learning method to obtain the decoder parameters.

Similar to ELM-AE [16], DAEF employs an auxiliary network to determine the parameters of each layer of the decoder in an unsupervised way, layer by layer. In the DAEF decoder, the weights and bias of a given $(l+1)$th hidden layer will be calculated with an auxiliary network, as will be explained. Finally, the output matrix of $(l+1)$th layer ($\mathbf{H}_{l+1}$) can be obtained as:

$$\mathbf{H}_{l+1} = f_{l+1}\left(\mathbf{W}_{l+1}^T\mathbf{H}_l + \mathbf{b}_{l+1}\mathbf{1}^T\right) \tag{10}$$

being f_{l+1} the activation function, $\mathbf{H}_l$ the output matrix of the lth layer with m_l neurons, $\mathbf{W}_{l+1} \in \mathbb{R}^{m_l\times m_{l+1}}$ and $\mathbf{b}_{l+1} \in \mathbb{R}^{m_{l+1}\times 1}$ the estimated weight matrix and bias vector of the layer, respectively, and $\mathbf{1}$ a column vector of n ones.

The use of the already mentioned auxiliary network is shown in the top right of Fig. 2, where $\mathbf{W}_{l+1}$ is obtained as the output weights of this network. As can be seen, the auxiliary network is a single-hidden layer sparse autoencoder. Regarding its structure, to calculate the parameters between the l and the $(l+1)$ hidden layers of the principal network, the number of neurons in the input and the output layers of the auxiliary network will be identical to m_l, the number of neurons of its hidden layer will be m_{l+1}, and f_{l+1} will be used as activation function.

The training of this auxiliary network can be divided into two stages: the training of the first half of the network, in which the input of the network $\mathbf{H}_{aux_0}$ (which is the output $\mathbf{H}_l$ of the DAEF's principal network lth layer) is transformed at the hidden layer; and a second stage, in which, using the data $\mathbf{H}_{aux_1}$ coming from the hidden layer, the original input is reconstructed at the output $\mathbf{H}_{aux_2}$ of the auxiliary network.

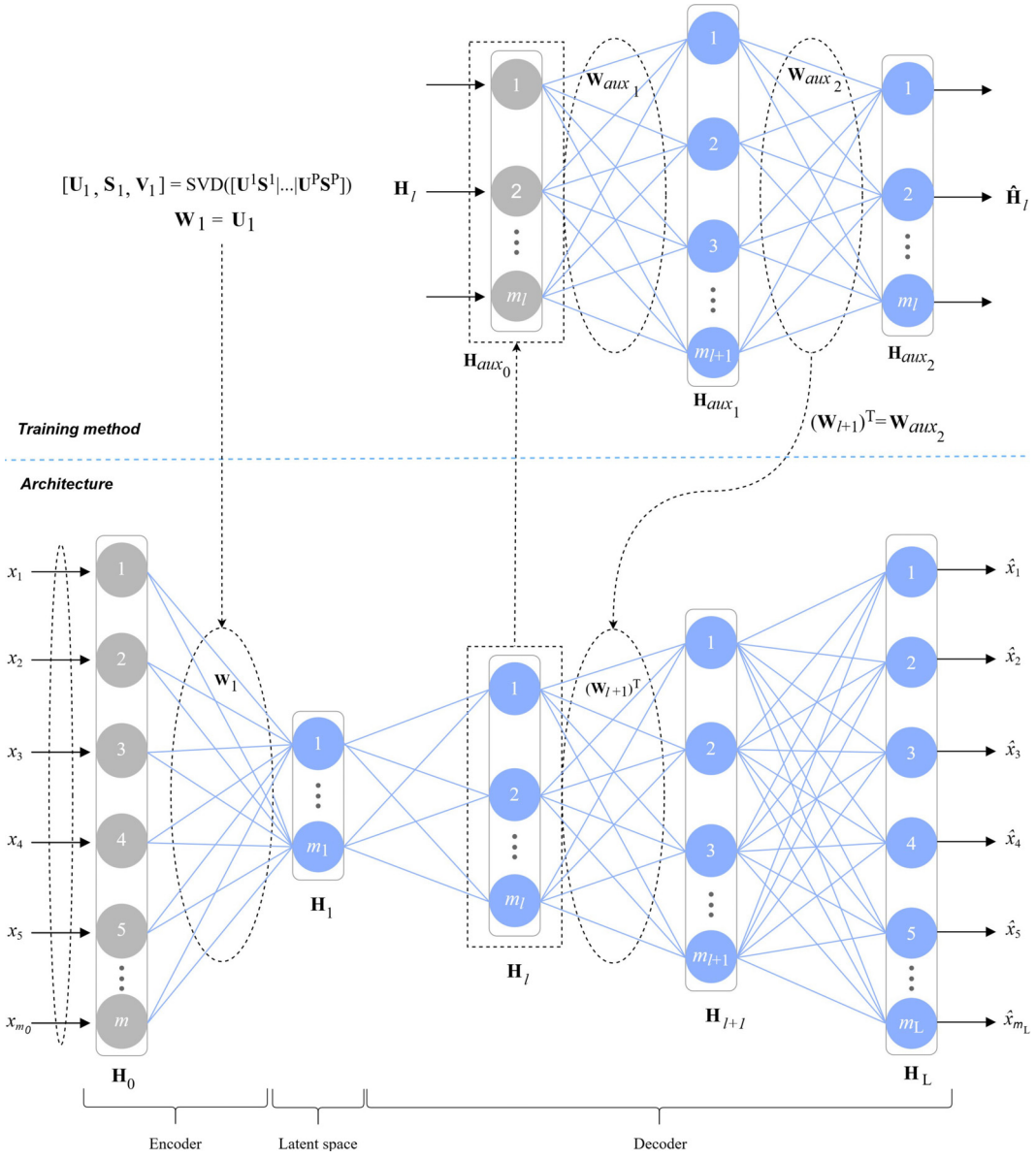

Fig. 2. The asymmetric deep autoencoder DAEF.

The weights $\mathbf{W}_{aux_1}$ and the bias $\mathbf{b}_{aux_1}$ of the first stage are initialized and fixed (the different initialization schemes will be studied in Section 6.2). Subsequently, the $\mathbf{H}_{aux_1}$ output of the hidden layer can be calculated as:

$$\mathbf{H}_{aux_1} = f\left(\mathbf{W}_{aux_1}^T \mathbf{H}_{aux_0} + \mathbf{b}_{aux_1}\mathbf{1}^T\right), \tag{11}$$

In the second stage, the weights $\mathbf{W}_{aux_2}$ are obtained in a supervised way using ROLANN (Algorithm 2). Considering that each output of the neural network depends solely on a set of independent weights, this second stage can be computed in parallel. Once this is calculated, the weights between the principal network's lth and $(l+1)$th layers can be obtained as $\mathbf{W}_{l+1} = \mathbf{W}_{aux_2}^T$, and the output $\mathbf{H}_{l+1}$ can be calculated using Eq. 10.

This process will be repeated for each of the hidden layers of the decoder, using the outputs of one layer to calculate the weights of the next one, until reaching the last layer. Finally, as the output target values for the DAEF's last layer are known (the same as in the DAEF's input layer), the weights of the last layer can be calculated directly in a supervised and distributed way using again ROLANN (Algorithm 2). We can summarize the DAEF training as follows:

1. Dimensionality reduction in the first layer using distributed SVD (encoder).
2. Unsupervised/supervised training, layer by layer, using an auxiliary network in which ROLANN is used as a regularization method (decoder).
3. Supervised training of the last layer using ROLANN (decoder).

4.3. Incremental, distributed, and parallel learning

DAEF performs various operations that can be computed in a parallel way if the node (device) on which it is executed has several cores. These operations are the SVD computation of the encoder (the dataset can be divided and the partial SVDs concatenated and recalculated) and the ROLANN regularization processes in the decoder (the weights concerning the output layer can be calculated in parallel).

In addition to this, the trained DAEF models can be updated when new data arrives thanks to their incremental learning capac-

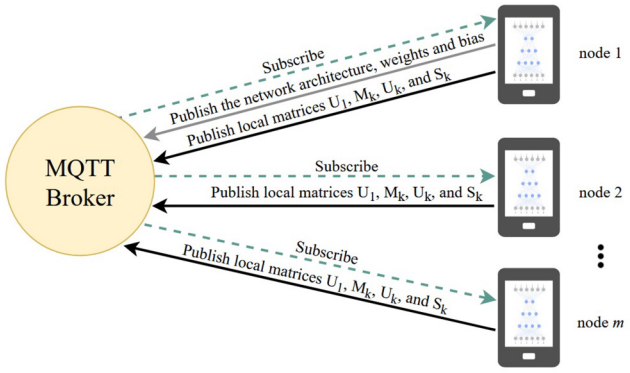

Fig. 3. DAEF networks collaborating through an MQTT protocol.

ity. A node can add knowledge to its model without having to retrain from scratch, incorporating the new knowledge quickly and inexpensively. A DAEF network trained with a data partition can incorporate the knowledge obtained by a second DAEF network trained with a different partition if the latter shares the $\mathbf{U}_{m_1}$ matrices of its encoder [21], and the $\mathbf{M}_k$, $\mathbf{U}_k$, and $\mathbf{S}_k$ matrices of each layer of its decoder [23]. By adding this information, the first DAEF network can recalculate its weights and will have learned incrementally.

If we are faced with a multi-node environment, such as an IoT scenario, where each node has a partition of the global dataset, we can take advantage of the incremental capacity of the DAEF network to carry out a distributed training. Each node (device) would train a DAEF autoencoder network with its local data, and using a protocol such as MQTT [25], these nodes can publish their local model information through a broker to share their particular knowledge with the rest of the devices. The broker will be in charge of sending this information to the nodes that are subscribed to the updates, which will be able to aggregate the information received to their model.

We consider the local dataset of each node as a partition of a global dataset, so all the nodes must use a DAEF autoencoder network with a similar architecture. For the model information shared between nodes to be compatible with each other, the nodes must also use the same weights generated by the Xavier Glorot initialization scheme and the same bias. At the start of the training, one of the nodes must define the architecture, generate the weights and bias and publish them through the broker. Fig. 3 shows this scenario using the MQTT protocol.

The private data of each node will be protected since the information that is sent through the broker is another. The data shared by each DAEF model (node) will be the $\mathbf{U}_{m_1}$ matrices of the encoder, and the $\mathbf{M}_k$, $\mathbf{U}_k$, and $\mathbf{S}_k$ matrices of each layer of the decoder, from which the original data are not recoverable [21,23]. The DAEF network matrices mentioned above are the only information needed to perform the federated learning, so if desired, the original dataset of each node can be removed to save space. Storing these matrices is not a problem since their size is independent of the number of instances of the original dataset.

Note that DAEF could also be used in a centralized scenario in which the information from the local models would be sent to a central node, which would be in charge of aggregating the information, obtaining the global model, and sharing it with the network nodes.

4.4. Pseudocode

Algorithm 3 contains the pseudocode for the DAEF training phase for one node with t available cores. The processes carried out in the encoder are described between lines 5 and 12. In line 7 the dimensionality of the data is reduced using SVD, in a parallel way splitting the local dataset into t partitions, obtaining the encoder weights and, in line 9, the encoder output. Between lines 13 and 20, the hidden layers of the decoder are trained one by one. For this, Algorithm 4 is used in line 15. In lines 21 and 22, the last layer of the decoder is trained directly using ROLANN also using parallelization.

Algorithm 3 DAEF training phase.

Input: $\mathbf{X} \in \mathbb{R}^{m_0 \times n}$, training dataset ($m_0$ variables $\times$ n samples); L, number of layers; a, list of number of neurons per layer; λ_{hid} and λ_{last}, regularization hyperparameters of the hidden and last layers; f_{hid} and f_{last}, activation functions of the hidden and last layers; t, available cores.

Output: *Model*, trained model composed of the weights W_{list} and bias b_{list}, the training output $\mathbf{H}_L$, and the $matrix_{list}$ needed for incremental learning, i.e., $\mathbf{U}_1$ and $\mathbf{S}_1$ matrices of the encoder and the $\mathbf{M}_l$, $\mathbf{U}_l$, and $\mathbf{S}_l$ matrices of each layer of the decoder.

1: **function** DAEF_TRAIN
2: $W_{list} = \varnothing$ ▷ Layer weight list
3: $H_{list} = \varnothing$ ▷ Layer output list
4: $matrices_{list} = \varnothing$ ▷ matrix list for incremental learning
5: ▷ Learning the encoder
6: $\mathbf{X}_{partitioned}$ = Split $\mathbf{X}$ in t partitions
7: $\mathbf{U}_1, \mathbf{S}_1 = DSVD(\mathbf{X}_{partitioned}, a[1])$ ▷ a[1], latent space dimension
8: $\mathbf{W}_1 = \mathbf{U}_1$ ▷ Weights of the *encoder*
9: $\mathbf{H}_1 = f_{hid}((\mathbf{W}_1)^T\mathbf{X})$
10: Append $\mathbf{W}_1$ to W_{list}
11: Append $\mathbf{H}_1$ to H_{list}
12: Append $[\mathbf{U}_1, \mathbf{S}_1]$ to $matrix_{list}$
13: ▷ Learning the decoder
14: **for** $l = 2.L - 1$ **do** ▷ $L-1$, decoder hidden layers
15: $\mathbf{W}_l, \mathbf{b}_l, \mathbf{H}_l, \mathbf{M}_l, \mathbf{U}_l, \mathbf{S}_l = TLD(H_{list}[l-1], a[l], \lambda_{hid}, f_{hid}, t)$ ▷ Call to Algorithm 2
16: Append $\mathbf{W}_l$ to W_{list}
17: Append $\mathbf{b}_l$ to b_{list}
18: Append $\mathbf{H}_l$ to H_{list}
19: Append $[\mathbf{M}_l, \mathbf{U}_l, \mathbf{S}_l]$ to $matrix_{list}$
20: **end for**
21: $pool$ =Pool(t) ▷ Pool of t processes
22: $\mathbf{W}_L, \mathbf{b}_L, \mathbf{M}_L, \mathbf{U}_L, \mathbf{S}_L = pool.map(ROLANN, (H_{list}[L-1], \mathbf{X}, \lambda_{last}, []))$ ▷ Parallel training of the last layer with ROLANN
23: $\mathbf{H}_L = f_{last}((\mathbf{W}_L)^T H_{list}[l-1])$
24: Append $\mathbf{W}_L$ to W_{list}
25: Append $\mathbf{b}_L$ to b_{list}
26: Append $[\mathbf{M}_L, \mathbf{U}_L, \mathbf{S}_L]$ to $matrix_{list}$
27: $Model = W_{list}, b_{list}, \mathbf{H}_L, matrix_{list}$
28: **end function**

Algorithm 4 contains the pseudocode of the auxiliary function used in Algorithm 3 to train the different hidden layers of the decoder using an auxiliary autoencoder. In lines 2 and 3, the weights and bias are generated respectively, while in line 4 the output of the hidden layer is computed. Between lines 5 and 7, the decoder weights and the output are calculated using ROLANN. Since the weights concerning each neuron of the output layer are calculated independently, the t available cores will be used to perform these calculations in parallel.

Algorithm 5 contains the pseudocode for the DAEF prediction phase where the trained network will reconstruct a test sample. Comparing the input value and its reconstruction after passing through the network (reconstruction error), it could be classified

Algorithm 4 Train one layer of the decoder (TLD).

Input: $\mathbf{H}_{l-1} \in \mathbb{R}^{m_l \times n}$ output of principal network's layer $l-1$ used as training data for this algorithm; m_l, number of neurons of the layer l; λ_{hid}, regularization hyperparameter of the hidden layer; f_{hid}, activation function of the layer; t, available cores;

Output: $\mathbf{W}_l$, weights of the layer l; $\mathbf{b}_l$ bias of the layer; $\mathbf{H}_l$, output of the layer l; and a list with the matrices needed for incremental learning, [**M**, **U**, **S**];

function TLD
  $\mathbf{W}_{aux_1} = Xavier(m_{l-1}, m_l)$ ▷ Initial weights
  $\mathbf{b}_{aux_1} = Random(m_l, 1)$ ▷ Initial bias
  $\mathbf{H}_{aux_1} = f_{hid}(\mathbf{W}_{aux_1}^T \mathbf{H}_{l-1} + \mathbf{b}_{aux_1} \mathbf{1}^T)$
  $pool$ =Pool(t) ▷ Pool of t processes
  $\mathbf{W}_l, \mathbf{b}_l, \mathbf{M}_l, \mathbf{U}_l, \mathbf{S}_l = pool.map(ROLANN, (\mathbf{H}_{aux_1}, \mathbf{H}_{l-1}, \lambda_{hid}, []))$ ▷ Parallel training of the layer
  $\mathbf{H}_l = f_{hid}(\mathbf{W}_l^T \mathbf{H}_{l-1} + \mathbf{b}_l \mathbf{1}^T)$
end function

Algorithm 5 DAEF prediction phase.

Input: $\mathbf{X} \in \mathbb{R}^{m_0 \times n}$, test dataset ($m_0$ variables $\times$ n samples); W_{list}, weights of the trained network; b_{list}, bias of the trained network; f_{hid} and f_{last}, activation functions of the hidden and last layers; a, list of number of neurons per layer;

Output: *prediction*, reconstruction of the input **X** after passing through the network;

function DAEF_PREDICT
  $\mathbf{H}_1 = f_{hid}((W_{list}[1])^T \mathbf{X})$
  for $l = 2.length(W_{list}) - 1$ **do**
    $\mathbf{H}_l = f_{hid}((W_{list}[l])^T \mathbf{H}_{l-1} + b_{list}[l-1]\mathbf{1}^T)$
  end for
  $prediction = f_{last}((W_{list}[l])^T \mathbf{H}_l + b_{list}[l]\mathbf{1}^T)$
end function

as normal or anomalous. The complete implementation of the algorithm is available in GitHub[1].

5. Privacy treatment

In distributed environments (EC and FL), preserving the privacy of data at each node is a critical aspect, even more so if they contain sensitive information, such as personal data. Due to this, in this section, we will analyze the privacy preservation capacity of the DAEF method. To do so we will consider two main threat scenarios [26].

5.1. Preventing direct leakage

In classic environments, it is common for the original data from the nodes to be sent to other nodes or to a central server, for example, to be analyzed, pre-processed, or to build a global machine learning model. This puts the privacy of the data at risk, which can be used maliciously and not to carry out the original tasks.

In the case of the DAEF method, the data shared for training the global model is not the original data (**X**). Regarding the encoder block, each node p, using its local data ($\mathbf{X}^p$), computes an $SVD = \mathbf{USV}^p$ and the only information shared to carry out the federated learning is the product $\mathbf{U}^p\mathbf{S}^p$. Since the matrix $\mathbf{V}^p$ is neither calculated nor sent, the original data $\mathbf{X}^p$ cannot be retrieved through the factorization expression described in Eq. 1. Regarding the decoder block, the federated learning is carried out by interchanging the $\mathbf{M}^p$, $\mathbf{U}^p$, and $\mathbf{S}^p$ matrices obtained through ROLANN regularization, so the original data is also kept safe. Once the global model is obtained, it is distributed to each of the local nodes p to be used privately, so there is also no direct data leakage in the operation phase.

5.2. Preventing indirect leakage

Another possible scenario is one in which a malicious node impersonates a real participant of the distributed learning protocol to try to obtain the private data of other nodes. Due to the nature, when we train models in a distributed way, it is common for nodes to share their calculations and parameters. Using this information, several authors have proposed specific methods that in certain cases can obtain the original training data, putting the privacy of the nodes at risk. The model inversion attack [27], once the network has been trained, follows the gradient used to adjust the weights of the network and obtains a reverse-engineered example for all represented classes in the model. Another way to address the problem is to train a Generative Adversarial Network (GAN) [28] in parallel to the attacked network using its gradients, so that it manages to extract information about classes of data that it does not know.

In the case of DAEF, the method is not iterative, so this type of attack is not a problem. The model parameters are calculated in a single step, so it is not possible to train GAN networks in this way. In addition, as we have seen previously, stochastic gradients are not shared (they are not used), so model inversion attacks are also not possible.

6. Evaluation

Although autoencoder networks have several uses, the main task for which DAEF has been designed is anomaly detection. In this section, several experiments are presented to show its behavior in this kind of scenario.

6.1. Experimental setup

DAEF emerges as a fast alternative to perform anomaly detection in environments where time can be a critical factor, such as edge computing and federated learning scenarios. Iterative approaches achieve high accuracy in anomaly detection, but in certain cases, their long training times may make them unsuitable for these environments. This study aims to analyze the performance achieved by DAEF compared to the adaptation of Online Sequential Extreme Learning Machine (OS-ELM) for federated learning [18], another non-iterative approach that uses a single hidden layer, as well as iterative single hidden layer autoencoders (SHL-AE) and iterative multiple hidden layer autoencoders (MHL-AE), models that follow a classical approach based on iterative learning. For this, a machine equipped with an Intel Core i7-11700k processor and 64GB of RAM has been used.

The algorithms were evaluated over seven datasets available in the UCI Machine Learning Repository [29], Kaggle [30], and ODDS [31]. Their characteristics are summarized in Table 1. All datasets were normalized using standard scalers with zero mean and unit variance. To assess the performance of each algorithm, the data were split using a 10-fold cross-validation. The algorithms were trained using only normal data, while the test phase included data from both classes (50% normal and 50% anomalies).

Given an autoencoder network trained with normal data, the classification of new instances can be carried out by comparing their value at the network input with the values obtained at the output. The difference is known as the reconstruction error, and

[1] https://github.com/DavidNovoaP/DAEF

Table 1
Characteristics of the datasets used. All represent anomaly detection data, except Covertype whose class 4 was taken as anomalous.

Dataset	Samples	Anomalies	Dimension
Shuttle	49,097	3511 (7.2%)	9
Covertype	581,012	2747 (1.0%)	10
Pendigits	6870	156 (2.3%)	16
Cardio	1831	176 (9.6%)	21
Credit card	284,807	492 (0.2%)	29
Ionosphere	351	126 (35.9%)	33
Optdigit	5216	64 (2.9%)	62

Table 2
Average test F1-score ± standard deviation for the different datasets.

Dataset	DAEF Ortho.	DAEF Random	DAEF Xavier
Shuttle	94.1±4.9	93.9±5.1	96.0±3.5
Covertype	81.4±6.3	80.1±9.7	86.0±4.4
Pendigits	70.2±9.3	70.6±12.1	76.3±5.5
Cardio	86.4±2.6	85.2±2.2	85.5±3.6
Credit card	89.1±1.8	87.6±2.3	90.0±0.6
Ionosphere	93.7±3.1	93.4±4.1	94.1±4.1
Optdigit	76.0±2.3	75.2±6.1	77.0±4.0

since anomalies are very rare in these scenarios, instances corresponding to the normal class will have a low reconstruction error, while anomalies will emit a much higher error. The reconstruction errors are calculated using the MSE (Mean Squared Error) and, after training the network, it is necessary to establish an error threshold above which input samples are classified as anomalies. There are many alternatives, a popular one being to define this threshold based on the interquartile range (IQR) of the reconstruction errors of the training examples, i.e.:

$$IQR = Q_3 - Q_1 \tag{12}$$

where Q_1 and Q_3 represent the first and the third quartiles. In this work, we define two error thresholds, one for outliers (*outlier IQR*) and another for extreme outliers (*extreme IQR*):

$$outlier\ IQR = Q_3 + 1.5 \times IQR \tag{13}$$

$$extreme\ IQR = Q_3 + 3 \times IQR \tag{14}$$

In addition, we also considered other thresholds using fixed percentiles (Q_{95}, Q_{90}, Q_{80}, Q_{70}, Q_{60} and Q_{50}).

Considering the anomalous class as the positive one, and based on the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), to measure the performance of the algorithms the F1-score metric was used:

$$F_1 = 2\frac{precision \cdot recall}{precision + recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{15}$$

Finally, the combinations of parameters chosen for each algorithm, as well as the error thresholds, were selected using a grid search and are available in Appendix A (Tables A1 and A2).

6.2. Choosing the best initialization for DAEF

Firstly, the effects of using different types of initialization for the weights and biases of the DAEF network were studied. Table 2 shows the results obtained by the DAEF model using three types of initializations: orthogonal, random, and Xavier Glorot. In all cases, we have used sigmoid activation functions in the hidden layers and linear functions in the outputs since we want to reconstruct in the output any real input data to the network.

As can be seen, the performance of DAEF is very similar for the three types of initializations, with Xavier Glorot being slightly higher in some cases, such as the Covertype dataset. To validate this statement, statistical tests were carried out to compare the global performance of the three approaches. The chosen procedure has been Kruskal-Wallis and Tukey's HSD as a post hoc test [32,33]. Using a significance level of 5% in both cases and the F1-scores of the algorithms for the different datasets, the Xavier Glorot initialization and the orthogonal initialization rank first, while random initialization is in a lower group, represented graphically by Fig. 4. This allows us to affirm that the performance of the DAEF model is equivalent using Xavier Glorot and orthogonal initializations, these being slightly better than the totally random one. This could be related to the difficulty of sigmoid activation functions to deal with small random weights [34]

Table 3
Average training time (seconds) ± standard deviation for the different datasets.

Dataset	DAEF Ortho.	DAEF Random	DAEF Xavier
Shuttle	1.8±0.1	1.8±0.1	1.8±0.1
Covertype	4.2±0.0	4.3±0.0	4.2±0.0
Pendigits	1.8±0.0	1.8±0.0	1.8±0.0
Cardio	1.9±0.0	1.9±0.0	1.9±0.0
Credit card	36.3±0.4	36.4±0.3	36.3±0.3
Ionosphere	1.9±0.0	1.8±0.0	1.9±0.0
Optdigit	3.4±0.0	3.4±0.0	3.6±0.1

Table 4
Average test F1-score ± standard deviation for the different datasets.

Dataset	DAEF	OS-ELM	SHL-AE	MHL-AE
Shuttle	96.0±3.5	**97.9±0.3**	**98.0±1.1**	**98.4±1.0**
Covertype	**86.0±4.4**	**85.8±0.2**	72.0±4.9	**81.5±13.3**
Pendigits	76.3±5.5	**88.2±2.0**	**84.2±4.4**	**86.3±6.1**
Cardio	**85.5±3.6**	**88.1±1.3**	**87.8±1.7**	**87.2±2.7**
Credit card	**90.0±0.6**	**90.7±0.5**	**91.0±0.6**	89.0±1.6
Ionosphere	**94.1±4.1**	**96.7±3.2**	91.9±2.4	**94.0±2.3**
Optdigit	77.0±4.0	81.7±0.8	83.1±7.7	**88.9±3.7**

Table 3 shows the mean training time of each algorithm (lower values than 0.05 have been represented as 0.0). Test times have not been included in this work because they are very low for all the algorithms. As can be seen, the training times are very similar regardless of the initialization used. Due to this, and his slightly better performance, Xavier Glorot will be considered the default initialization for DAEF and will be used in the remaining tests.

6.3. DAEF vs. other Anomaly Detection algorithms

In this section, we compare the performance obtained by DAEF (Xavier Glorot initialization) and the algorithms mentioned in Section 6.1 (OS-ELM, SHL-AE, and MHL-AE). Since the goal is to measure only their ability to detect anomalies, DAEF and OS-ELM were executed using a single node. Notice that running the OS-ELM adaptation for FL [18] using a single node is equivalent to using the original OS-ELM algorithm [19].

Table 4 summarizes the mean test results. Again, Kruskal-Wallis and Tukey's HSD tests (significance level of 5%) have been used for each dataset to highlight in bold the models that rank first. As can be seen, the DAEF algorithm achieves good performance for most datasets, matching the performance of the best models in four of the seven datasets (Covertype, Cardio, Credit card, and Ionosphere), and getting a very close performance on datasets such as Shuttle. A Nemenyi statistical test [35,36] was carried out to compare the global performance of the algorithms. Using a significance level of 5% and the F1-scores of the algorithms for the different datasets, the four methods rank in the same position, represented graphically by Fig. 5.

Table 5 shows the mean training time of each algorithm. Because DAEF training is non-iterative, its training times are much shorter than those required by traditional iterative autoencoders

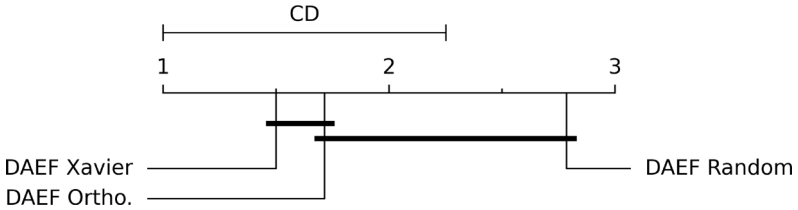

Fig. 4. Graphical representation of Nemenyi test with $\alpha = 0.05$. The critical distance (CD) obtained was 1.25.

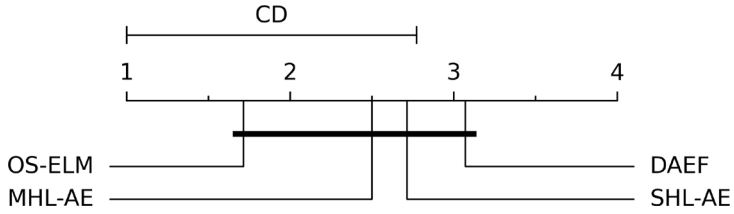

Fig. 5. Graphical representation of the Nemenyi test with $\alpha = 0.05$. The critical distance (CD) obtained was 1.77.

Table 5
Average training time (seconds) ± standard deviation for the different datasets.

Dataset	DAEF	OS-ELM	SHL-AE	MHL-AE
Shuttle	1.8±0.1	0.0±0.0	53.9±1.0	157.8 ±1.7
Covertype	4.2±0.0	0.2±0.0	157.1±1.3	239.6±2.3
Pendigits	1.8±0.0	0.0±0.0	19.6 ± 2.2	58.6±4.0
Cardio	1.9±0.0	0.0±0.0	11.9±0.1	2.4±0.1
Credit card	36.3±0.3	0.9±0.0	387.2±6.2	194.7±3.6
Ionosphere	1.9±0.0	0.0±0.0	4.0±0.1	2.59±0.2
Optdigit	3.6±0.1	0.0±0.0	3.9 ±0.4	13.04±0.4

Table 6
Estimated emissions (CO_2-eq) for the different datasets.

Dataset	DAEF	OS-ELM	SHL-AE	MHL-AE
Shuttle	1.4×10^{-6}	2.8×10^{-7}	7.9×10^{-4}	2.5×10^{-3}
Covertype	3.3×10^{-5}	3.0×10^{-6}	2.2×10^{-3}	3.7×10^{-3}
Pendigits	5.0×10^{-6}	2.3×10^{-7}	2.9×10^{-4}	8.5×10^{-4}
Cardio	5.3×10^{-6}	7.9×10^{-8}	1.7×10^{-4}	3.1×10^{-5}
Credit card	4.2×10^{-4}	1.2×10^{-5}	5.3×10^{-3}	2.9×10^{-3}
Ionosphere	4.4×10^{-6}	3.5×10^{-8}	5.1×10^{-5}	3.7×10^{-5}
Optdigit	6.9×10^{-5}	2.7×10^{-7}	5.6×10^{-5}	1.9×10^{-4}
Mean	7.7×10^{-5}	2.3×10^{-6}	1.3×10^{-3}	1.5×10^{-3}

Table 7
Estimated energy consumed (kWh) for the different datasets.

Dataset	DAEF	OS-ELM	SHL-AE	MHL-AE
Shuttle	1.7×10^{-6}	4.7×10^{-7}	1.3×10^{-3}	4.1×10^{-3}
Covertype	5.3×10^{-5}	5.0×10^{-6}	3.7×10^{-3}	6.2×10^{-3}
Pendigits	8.5×10^{-6}	3.9×10^{-7}	4.8×10^{-4}	1.4×10^{-3}
Cardio	8.9×10^{-6}	1.3×10^{-7}	2.8×10^{-4}	5.3×10^{-5}
Credit card	7.0×10^{-4}	2.0×10^{-5}	8.9×10^{-3}	4.9×10^{-3}
Ionosphere	7.4×10^{-6}	5.9×10^{-8}	8.6×10^{-5}	6.2×10^{-5}
Optdigit	1.1×10^{-4}	4.6×10^{-7}	9.4×10^{-5}	3.3×10^{-4}
Mean	1.3×10^{-4}	3.8×10^{-6}	2.1×10^{-3}	2.4×10^{-3}

(SHL-AE and MHL-AE). However, it fails to outperform OS-ELM which is extremely fast. This difference in training time is due, in part, to the fact that OS-ELM uses a single hidden layer, while DAEF uses several.

Finally, Tables 6 and 7 show an estimation of carbon dioxide emissions (CO_2-eq) and energy consumption (kWh) for the machine on which the tests were run [37]. As can be seen, in most cases the consumption and emissions of DAEF are much lower than those of the iterative autoencoders (SHL-AE and MHL-AE) although, as expected, higher than those of OS-ELM.

6.4. DAEF in Federated Learning scenarios

To test the behavior of DAEF in federated environments, several experiments were carried out. The objective was to study its performance, training time, and energy consumption as a function of the number of nodes. The results were compared with the reference method OS-ELM [18]. Given a simulated FL environment with n nodes with similar computational resources, and a dataset, it is divided evenly into n partitions of equal size. Each of these nodes train a DAEF network with its local dataset partition, and then adds its local model to the global one. To ensure that the partitions are made up of a sufficiently large number of instances, the datasets chosen were the three datasets with the largest number of instances: Shuttle, Covertype, and Credit card. Training time was measured as the sum of the time needed by the slowest node during local training and the time of all aggregations performed during incremental learning.

Fig. 6 represents the average F1-score obtained using a different number of nodes (between 1 and 50). As can be seen, the performance of both models remains similar.

However, a common scenario in FL is one in which, either due to the existence of a very high number of nodes, or due to the scarcity of instances in the dataset, the number of instances per node is low. To test the DAEF and OS-ELM methods in this type of scenario, we used all the datasets mentioned in this article (Table 1). For each of these datasets, the F1-score of both methods was measured based on the number of nodes used, in a similar way to the previous experiments, but using a sufficiently high number of nodes to cause the number of instances assigned to each node to be low (~50 instances per node). Taking this into account, we have observed that the performance of the DAEF model remains stable regardless of the number of data per node, while the OS-ELM method tends to generate bad results when the number of data goes below a certain value (depending on the dataset). Table 8 shows, for each dataset, the conditions from which the performance of OS-ELM starts to degrade significantly. Except the Covertype dataset, where none of the models degrades its performance, we can state that, in general, it is remarkable how OS-ELM

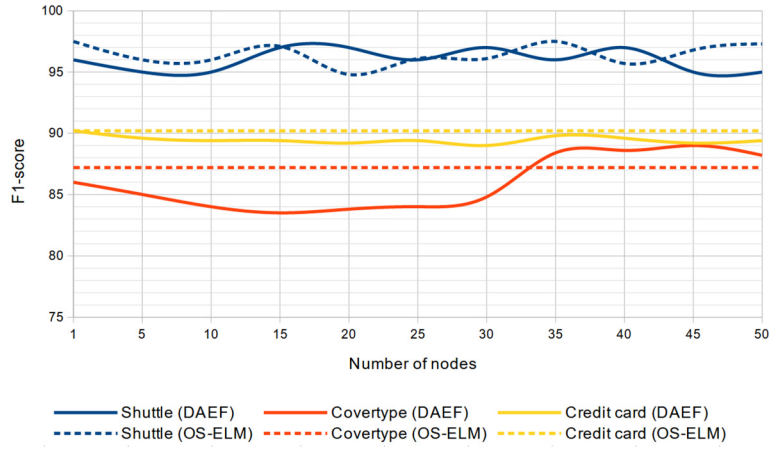

Fig. 6. Average F1-score of the final global model as a function of the number of nodes used in a federated scenario with DAEF and OS-ELM.

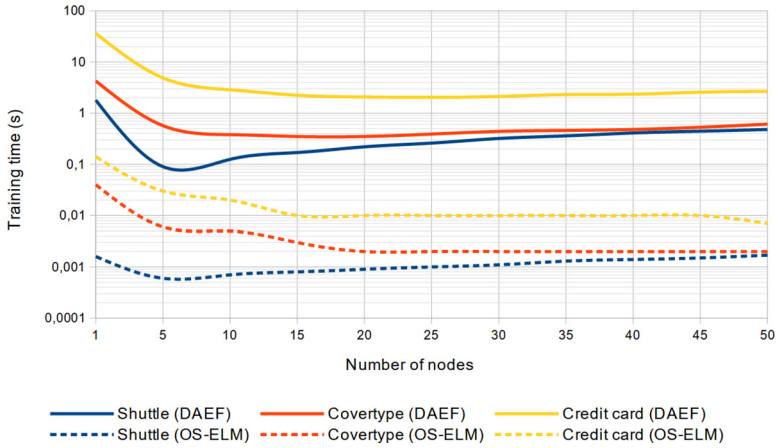

Fig. 7. Average training time (seconds) required by DAEF and OS-ELM in a federated learning scenario based on the number of nodes used.

Table 8
DAEF vs. OS-ELM (F1-score) when the amount of samples per node is extremely low. The "samples per node" column shows the critical point at which the OS-ELM model exhibits a significant drop in performance. The "DAEF" and "OS-ELM" columns contain the F1-score reached at that point.

Dataset	Nodes	Samples per node	DAEF	OS-ELM
Shuttle	1000	~50	96.1	86.2
Covertype	-	-	-	-
Pendigits	100	~60	76.2	60.5
Cardio	100	~14	86.1	74.1
Credit Card	8000	~31	89.5	71.2
Ionosphere	3	~67	85.7	53.5
Optdigit	100	~46	75.5	77.9

presents difficulties to learn in a federated way when the number of instances per node is extremely low (between 14 and 60), while DAEF maintains its original performance.

Regarding the training times, Fig. 7 contains the sum of the local training time and the aggregation of local models when using between 1 and 50 nodes. In both methods, the time consumed using multiple nodes is smaller than the time required for a single node centrally trained on the entire dataset. This is because the partitioning of the data set greatly accelerates local training, making most of the time required to complete training associated with the aggregation process. Even though the times of the federated version are better, we can observe a slight increase in training time as the number of nodes grows. This is due to the aggregation process performed by the coordinator that, although being a not very complex operation, scales linearly with the number of nodes used. Compared to the OS-ELM method, the latter is much faster, largely due to the use of a single hidden layer. Despite this, DAEF is fast enough for FL scenarios.

Finally, Fig. 8 shows the estimated energy consumption (kWh) during the training taking into account all the nodes involved (again between 1 and 50 nodes). Regarding DAEF's behavior, in the case of Shuttle, since it is the smaller dataset, the aggregation of models is the process with the greatest weight in total consumption, which justifies its marked increasing trend concerning the number of nodes. In the case of Covertype and Credit card, consumption shows slower growth in the initial stages because, for the same reasons, the computational cost of local training has a greater weight. Compared to the OS-ELM method, and under the training times already observed in Fig. 7, it is natural that the energy consumption of DAEF is higher. Again, this does not prevent its use in FL scenarios as it is still a very competitive consumption.

6.5. DAEF vs. Non-IID data

In horizontal federated learning scenarios, when the training data is not independent and identically distributed (Non-IID), the models usually perform worse than centralized alternatives [38]. To demonstrate that DAEF allows obtaining a global model equiva-

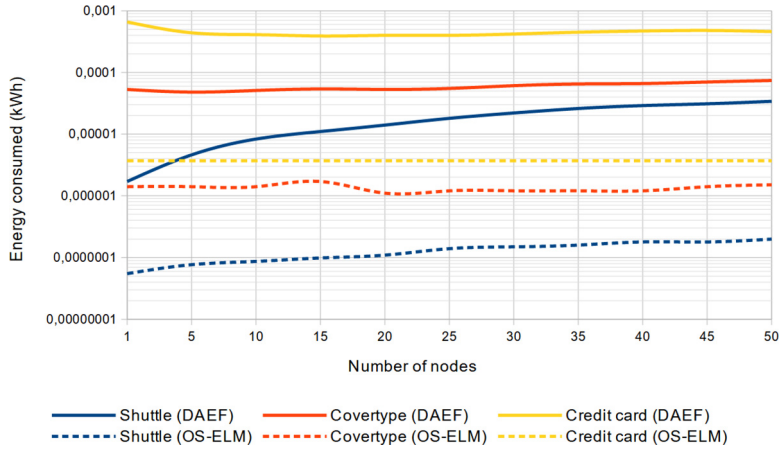

Fig. 8. Estimated energy consumption (kWh) during the training of DAEF and OS-ELM in a federated scenario based on the number of nodes.

Table 9
Average test F1-score ± standard deviation for the Covertype dataset considering each class as the anomalous and two types of training: IID centralized and Non-IID decentralized.

Anomalous class	IID Centralized	Non-IDD Decentralized
Class 1	75.0±3.1	75.2±2.6
Class 2	69.1±5.8	67.9±4.0
Class 3	91.5±3.2	90.3±3.6
Class 4	86.0±4.4	85.2±5.1
Class 5	82.4±3.9	81.8±4.6
Class 6	87.2±4.3	87.3±3.3
Class 7	79.06±5.1	80.2±2.3

lent to training with all the data in a centralized way also in Non-IID scenarios, a test has been carried out.

For this, the Covertype dataset has been used, which has seven different classes. Considering one of them as anomalous and the rest as normal, a non-IID scenario with no overlap between classes has been simulated. Simulating six nodes, each one has been trained with the instances corresponding to one of the six classes considered as normal. After this, the local models have been added, giving rise to a global model. The performance of this global model has been compared with the performance of a second model trained with all the normal data at once in a centralized way (IID). This test has been repeated ten times using a 10-fold. As can be seen in Table 9, the performance of both types of training was similar.

7. Conclusion

An alternative method to traditional deep autoencoder networks has been presented. Its distributed, parallel and incremental learning capability, its low computational cost and its privacy preservation make it a valid solution for edge computing and federated learning environments. For these scenarios, this paper has proposed a possible decentralized architecture to implement DAEF using MQTT as the communication protocol. Moreover, although the strengths of the method lie in its usefulness in edge computing and federated learning scenarios, it is also an interesting solution for centralized classical machine learning scenarios.

To prove this claim, DAEF has been compared with traditional approaches as well as with OS-ELM, a very efficient federated autoencoder. In all cases, DAEF shows a fairly competitive performance, with training times and energy consumption much lower than traditional iterative approaches. Regarding OS-ELM, its efficiency is hardly surpassable, however, DAEF has shown better performance when the number of instances per node is low and also allows the use of deep architectures, which positions it as a good alternative for edge computing and federated learning scenarios that require more complex models, a context for which there are currently not many solutions.

As future work, it would be interesting to test the algorithm in real edge computing or federated learning environments using real physical devices that act as independent nodes, instead of simulating them. Another possible line of work would be to adapt the method to scenarios in which the distribution of the normal class changes over time (distribution shift), implementing, for example, a forgetting mechanism.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by grant Machine Learning on the Edge - Ayudas Fundación BBVA a Equipos de Investigación Científica 2019; the Spanish National Plan for Scientific and Technical Research and Innovation (PID2019-109238GB-C22 and TED2021-130599A-I00); the Xunta de Galicia (ED431C 2022/44) and ERDF funds. CITIC is funded by Xunta de Galicia and ERDF funds. Funding for open access charge: Universidade da Coruña/CISUG.

Appendix A. Parameters used during training

This appendix contains the values of the parameters finally chosen as the best for each method and dataset, listed in Tables A1 and A2.

Table A1
Parameters used during the experimentation.

Dataset	DAEF Ortho.	DAEF Random	DAEF Xavier
Shuttle	Arch: [9, 5, 7, 9], λ_{hid}: 0.01, λ_{last}: 0.8, μ: outlier IQR	Arch: [9, 5, 7, 9], λ_{hid}: 0.8, λ_{last}: 0.8, μ: outlier IQR	Arch: [9, 5, 7, 9], λ_{hid}: 0.8, λ_{last}: 0.3, μ: outlier IQR
Covertype	Arch: [10, 6, 8, 10], λ_{hid}: 0.1, λ_{last}: 0.9, μ: P_{80}	Arch: [10, 6, 8, 10], λ_{hid}: 0.05, λ_{last}: 0.1, μ: P_{80}	Arch: [10, 6, 8, 10], λ_{hid}: 0.005, λ_{last}: 0.8, μ: P_{80}
Pendigits	Arch: [16, 5, 10, 16], λ_{hid}: 0.3, λ_{last}: 0.8, μ: P_{40}	Arch: [16, 5, 10, 16], λ_{hid}: 0.01, λ_{last}: 0.8, μ: P_{40}	Arch: [16, 5, 10, 16], λ_{hid}: 0.8, λ_{last}: 0.3, μ: P_{60}
Cardio	Arch: [21, 10, 15, 21], λ_{hid}: 0.9, λ_{last}: 0.9, μ: P_{90}	Arch: [21, 10, 15, 21], λ_{hid}: 0.9, λ_{last}: 0.7, μ: P_{90}	Arch: [21, 10, 15, 21], λ_{hid}: 0.9, λ_{last}: 0.2, μ: P_{80}
Credit card	Arch: [29, 15, 20, 25, 29], λ_{hid}: 0.9, λ_{last}: 0.7, μ: outlier IQR	Arch: [29, 15, 20, 25, 29], λ_{hid}: 0.005, λ_{last}: 0.9, μ: P_{90}	Arch: [29, 15, 20, 25, 29], λ_{hid}: 0.01, λ_{last}: 0.9, μ: outlier IQR
Ionosphere	Arch: [33, 20, 25, 33], λ_{hid}: 0.005, λ_{last}: 0.9, μ: outlier IQR	Arch: [33, 20, 25, 33], λ_{hid}: 0.01, λ_{last}: 0.8, μ: outlier IQR	Arch: [33, 20, 25, 33], λ_{hid}: 0.005, λ_{last}: 0.8, μ: P_{90}
Optdigit	Arch: [62, 20, 30, 40, 50, 62], λ_{hid}: 0.01, λ_{last}: 0.8, μ: P_{40}	Arch: [62, 30, 40, 50, 62], λ_{hid}: 0.01, λ_{last}: 0.9, μ: P_{50}	Arch: [62, 20, 30, 40, 50, 62], λ_{hid}: 0.01, λ_{last}: 0.3, μ: P_{40}

Table A2
Parameters used during the experimentation.

Dataset	OS-ELM	SHL-AE	MHL-AE
Shuttle	Arch: [9, 7, 9], Batch: 100, μ: extreme IQR	Arch: [9, 5, 9] Epochs: 100, μ: extreme IQR	Arch: [9, 7, 5, 3, 5, 7, 9] Epochs: 200, μ: extreme IQR
Covertype	Arch: [10, 8, 10], Batch: 500, μ: P_{80}	Arch: [10, 6, 10] Epochs: 10, μ: P_{70}	Arch: [10, 8, 6, 4, 6, 8, 10] Epochs: 10, μ: P_{90}
Pendigits	Arch: [16, 12, 16], Batch: 100, μ: P_{80}	Arch: [16, 8, 16] Epochs: 50, μ: P_{80}	Arch: [16, 12, 8, 4, 8, 12, 16] Epochs: 100, μ: outlier IQR
Cardio	Arch: [21, 5, 21], Batch: 100, μ: P_{80}	Arch: [21, 5, 21] Epochs: 100, μ: P_{90}	Arch: [21, 15, 10, 5, 10, 15, 21] Epochs: 90, μ: P_{90}
Credit card	Arch: [29, 25, 29], Batch: 100, μ: extreme IQR	Arch: [29, 10, 29] Epochs: 25, μ: outlier IQR	Arch: [29, 20, 10, 20, 29] Epochs: 10, μ: extreme IQR
Ionosphere	Arch: [33, 20, 33], Batch: 100, μ: extreme IQR	Arch: [33, 15, 33] Epochs: 100, μ: outlier IQR	Arch: [33, 25, 20, 15, 20, 25, 33] Epochs: 50, μ: extreme IQR
Optdigit	Arch: [62, 20, 62], Batch: 100, μ: P_{60}	Arch: [62, 30, 62] Epochs: 10, μ: P_{80}	Arch: [62,50, 40, 30, 20, 30, 40,50, 62] Epochs: 25, μ: P_{90}

References

[1] K. Cao, Y. Liu, G. Meng, Q. Sun, An overview on edge computing research, IEEE Access 8 (2020) 85714–85728, doi:10.1109/ACCESS.2020.2991734.

[2] Y. Zhang, Y. Xu, S. Wei, Y. Wang, Y. Li, X. Shang, Doubly contrastive representation learning for federated image recognition, Pattern Recognit 139 (2023) 109507, doi:10.1016/j.patcog.2023.109507.

[3] G. Pang, C. Shen, L. Cao, A.V.D. Hengel, Deep learning for anomaly detection: a review, ACM Comput. Surv. 54 (2) (2021), doi:10.1145/3439950.

[4] R. Domingues, M. Filippone, P. Michiardi, J. Zouaoui, A comparative evaluation of outlier detection algorithms: experiments and analyses, Pattern Recognit 74 (2018) 406–421, doi:10.1016/j.patcog.2017.09.037.

[5] W.L. Al-Yaseen, A.K. Idrees, F.H. Almasoudy, Wrapper feature selection method based differential evolution and extreme learning machine for intrusion detection system, Pattern Recognit 132 (2022) 108912, doi:10.1016/j.patcog.2022.108912.

[6] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. 41 (3) (2009), doi:10.1145/1541880.1541882.

[7] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, 2021, 2003.05991

[8] T.D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, A.-R. Sadeghi, DÏot: A federated self-learning anomaly detection system for iot, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 756–767, doi:10.1109/ICDCS.2019.00080.

[9] B. Hussain, Q. Du, S. Zhang, A. Imran, M.A. Imran, Mobile edge computing-based data-driven deep learning framework for anomaly detection, IEEE Access 7 (2019) 137656–137667, doi:10.1109/ACCESS.2019.2942485.

[10] R.A. Sater, A.B. Hamza, A federated learning approach to anomaly detection in smart buildings, ACM Trans. Internet Things 2 (4) (2021), doi:10.1145/3467981.

[11] Y. Zhao, J. Chen, D. Wu, J. Teng, S. Yu, Multi-task network anomaly detection using federated learning, in: SoICT 2019, ACM, 2019, pp. 273–279, doi:10.1145/3368926.3369705.

[12] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, E. Ilie-Zudor, Chained anomaly detection models for federated learning: an intrusion detection case study, Appl. Sci. 8 (12) (2018), doi:10.3390/app8122663.

[13] T. Luo, S.G. Nagarajan, Distributed anomaly detection using autoencoder neural networks in WSN for IoT, in: IEEE ICC, 2018, pp. 1–6, doi:10.1109/ICC.2018.8422402.

[14] M.V. Ngo, T. Luo, T.Q.S. Quek, Adaptive anomaly detection for internet of things in hierarchical edge computing: a contextual-bandit approach, ACM Trans. Internet Things 3 (1) (2021), doi:10.1145/3480172.

[15] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501, doi:10.1016/j.neucom.2005.12.126. Neural Networks

[16] L. Kasun, H. Zhou, G.-B. Huang, C.-M. Vong, Representational learning with ELMs for Big Data, IEEE Intell Syst 28 (2013) 31–34.

[17] S. Ding, N. Zhang, X. Xu, L. Guo, J. Zhang, Deep extreme learning machine and its application in EEG classification, Math. Probl. Eng. 2015 (2015) 1–11, doi:10.1155/2015/129021.

[18] R. Ito, M. Tsukada, H. Matsutani, An on-device federated learning approach for cooperative model update between edge devices, IEEE Access 9 (2021), doi:10.1109/access.2021.3093382. 92986-92998

[19] N.-y. Liang, G.-b. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, IEEE Trans. Neural Networks 17 (6) (2006) 1411–1423, doi:10.1109/TNN.2006.880583.

[20] H. Zhang, J. Bosch, H.H. Olsson, Federated learning systems: architecture alternatives, in: 2020 27th Asia-Pacific Software Engineering Conference (APSEC), 2020, pp. 385–394, doi:10.1109/APSEC51365.2020.00047.

[21] O. Fontenla-Romero, B. Pérez-Sánchez, B. Guijarro-Berdiñas, DSVD-autoencoder: a scalable distributed privacy-preserving method for one-class classification, Int. J. Intell. Syst. 36 (1) (2021) 177–199, doi:10.1002/int.22296.

[22] C. Eckart, G. Young, The approximation of one matrix by another of lower rank, Psychometrika 1 (3) (1936) 211–218, doi:10.1007/BF02288367.

[23] O. Fontenla-Romero, B. Guijarro-Berdiñas, B. Pérez-Sánchez, Regularized one-layer neural networks for distributed and incremental environments, in: IWANN, volume 12862, Springer, 2021, pp. 343–355, doi:10.1007/978-3-030-85099-9_28.

[24] O. Fontenla-Romero, B. Guijarro-Berdiñas, B. Pérez-Sánchez, A. Alonso-Betanzos, A new convex objective function for the supervised learning of single-layer neural networks, Pattern Recogn. 43 (5) (2010) 1984–1992, doi:10.1016/j.patcog.2009.11.024.

[25] B. Mishra, A. Kertesz, The use of MQTT in M2M and IoT systems: asurvey, IEEE Access 8 (2020) 201071–201086, doi:10.1109/ACCESS.2020.3035849.

[26] V. Mothukuri, R.M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, G. Srivastava, A survey on security and privacy of federated learning, Future Generation Computer Systems 115 (2021) 619–640, doi:10.1016/j.future.2020.10.007.

[27] K.-C. Wang, Y. FU, K. Li, A. Khisti, R. Zemel, A. Makhzani, Variational model inversion attacks, in: Advances in Neural Information Processing Systems, volume 34, Curran Associates, 2021, pp. 9706–9719.

[28] B. Hitaj, G. Ateniese, F. Perez-Cruz, Deep models under the GAN: Information leakage from collaborative deep learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, in: CCS '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 603–618, doi:10.1145/3133956.3134012.

[29] D. Dua, C. Graff, UCI machine learning repository, 2017, http://archive.ics.uci.edu/ml

[30] Worldline, the ML Group of ULB, Credit card fraud detection, 2014, https://www.kaggle.com/mlg-ulb/creditcardfraud

[31] S. Rayana, ODDS library, 2016, http://odds.cs.stonybrook.edu.

[32] E. Ostertagova, O. Ostertag, J. Kovác, Methodology and application of the kruskal-wallis test, Applied Mechanics and Materials 611 (2014) 115–120.

[33] A. Nanda, A. Mahapatra, B. Mohapatra, a. mahapatra, Multiple comparison test by tukey's honestly significant difference (hsd): do the confident level control type i error, International Journal of Applied Mathematics and Statistics 6 (2021) 59–65, doi:10.22271/maths.2021.v6.i1a.636.

[34] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[35] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (1) (2006) 1–30.

[36] S. García, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, J. Mach. Learn. Res. 9 (89) (2008) 2677–2694.

[37] V. Schmidt, K. Goyal, A. Joshi, B. Feld, L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, S. Luccioni, CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing (2021). 10.5281/zenodo.4658424

[38] H. Zhu, J. Xu, S. Liu, Y. Jin, Federated learning on non-iid data: a survey, Neurocomputing 465 (2021) 371–390, doi:10.1016/j.neucom.2021.07.098.

David Novoa-Paradela (M) received his B.S. degree in Computer Science from the University of A Coruña in 2019, and his M.S. degree in Research in Artificial Intelligence from the Menendez Pelayo International University in 2020. In October 2020 he started his Ph.D. thesis focused on the field of Anomaly Detection.

Oscar Fontenla-Romero (M) Ph.D. in Computer Science and Full Professor in Artificial Intelligence at the University of A Coruña. His research has focused on the development of new machine learning models, as well as its application in engineering and biomedicine areas. He has been part of the Board of Directors of the Spanish Association for Artificial Intelligence (AEPIA) from 2013 to 2018.

Bertha Guijarro-Berdiñas (F) Ph.D. in Computer Science and Associate Professor at the University of A Coruña. Her research interests focus on Artificial Intelligence with special attention to the theoretical aspects of machine learning (distributed, online, scalable, sustainable and efficient learning, privacy preservation) and its applications. She has participated in more than 30 national and international projects, agreements with companies and is co-author of more than 100 articles.