Feature Encoding with Autoencoder and Differential Evolution for Network Intrusion Detection using Machine Learning

Miguel Leon*
miguel.leonortiz@mdu.se
School of Innovation, Design and Engineering, Malardalen University
Vasteras, Sweden

Tijana Markovic*
tijana.markovic@mdu.se
School of Innovation, Design and Engineering, Malardalen University
Vasteras, Sweden

Sasikumar Punnekkat
sasikumar.punnekkat@mdu.se
School of Innovation, Design and Engineering, Malardalen University
Vasteras, Sweden

ABSTRACT

With the increasing use of computer networks and distributed systems, network security and data privacy are becoming major concerns for our society. In this paper, we present an approach based on an autoencoder trained with differential evolution for feature encoding of network data with the goal of improving security and reducing data transfers. One of the novel elements used in differential evolution for intrusion detection is the enhancements in the fitness function by adding the performance of a machine learning algorithm. We conducted an extensive evaluation of six machine learning algorithms for network intrusion detection using encoded data from well-known publicly available network datasets UNSW-NB15. The experiments clearly showed the supremacy of random forest, support vector machine, and K-nearest neighbors in terms of accuracy, and this was not affected to a high degree by reducing the number of features. Furthermore, the machine learning algorithm that was used during training (Linear Discriminant Analysis classifier) got a 14 percentage points increase in accuracy. Our results also showed clear improvements in execution times in addition to the obvious secure aspects of encoded data. Additionally, the performance of the proposed method outperformed one of the most commonly used feature reduction methods, Principal Component Analysis.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Neural networks**; **Bio-inspired approaches**; *Supervised learning*; *Unsupervised learning*.

KEYWORDS

Differential Evolution, Neural Networks, Autoencoder, Machine Learning, Intrusion Detection

ACM Reference Format:
Miguel Leon, Tijana Markovic, and Sasikumar Punnekkat. 2022. Feature Encoding with Autoencoder and Differential Evolution for Network Intrusion Detection using Machine Learning. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3520304.3534009

*Both authors contributed equally to this research.

1 INTRODUCTION

As our society is becoming increasingly dependent on the products and services provided through connectivity to digital infrastructures, there is also an increase in the types, incidences, and complexity of cyber-attacks on our networks. New and more sophisticated cyber security threats are being discovered on a daily basis, and security measures must be constantly updated and improved. Identifying these threats on time is one of the biggest challenges for network security, and Intrusion Detection (ID) is one of commonly used techniques. It aims to identify unauthorized use, misuse and abuse of computer systems by internal and external threats [31].

Different Machine Learning (ML) algorithms have been successfully used in ID during recent years, resulting in high detection rates and good overall performance [38]. ML is part of artificial intelligence in which algorithms use data to automatically improve through experience [28]. ML algorithms have the ability to learn and the learning process can be supervised, unsupervised, and reinforcement. Supervised Learning (SL) and Unsupervised Learning (UL) methods are used for ID. The difference between these two learning technologies is the existence of labels in the training data subset [6]. SL algorithms use input attributes and the desired output to learn a function that maps from inputs to outputs, whereas UL algorithms use only input attributes to learn patterns. From an ML perspective, searching for intrusions can be seen as a classification problem. Intrusion detection is a binary classification problem in which the algorithm should learn to distinguish between normal and malicious activities. If there is a need not only to detect malicious activity, but also to determine which malicious activity occurred, then it is a multiclass classification problem. Various ML algorithms have been evaluated on different ID datasets and have proven to be very successful. Most commonly used ML algorithms are Random Forest (RF), Decision Trees, Support Vector Machine (SVM), K-Nearest Neighbor (K-NN), Linear Discriminant Analysis classfier (LDA), Artificial Neural Networks (ANN), Naive Bayes (NB), density-based clustering algorithms (e.g., DBSCAN), K-means, etc. [12, 23].

Network datasets are collected by recording network traffic in the form of packets that can have a large number of features. Some of those features can be redundant or irrelevant for intrusion detection. In addition, big datasets with a large number of features can have a negative influence on detection speed and computational costs [32]. Usually, different feature reduction techniques are performed to improve learning capabilities and reduce the computational intensity of ML models [5, 8, 10].

In this paper, we propose the use of an autoencoder trained by differential evolution for feature reduction and encoding of network data. Differential Evolution (DE) is a population-based optimization algorithm that has been used in many different applications, ranging from the optimization of harmonic filter design [25] to intrusion detection [3]. The proposed approach not only reduces the number of features, but also encodes the features, which is beneficial to preserve the privacy of the data. Experiments were conducted on the well-known ID dataset UNSW-NB15. We evaluated five supervised and one unsupervised ML algorithms and compared their performance before and after the feature encoding process. Furthermore, we compared this method with one of the most widely used techniques for feature reduction, Principal Component Analysis (PCA) [2].

Multiple studies [4, 7, 14, 18, 22, 27] presented the use of autoencoders for ID, but the proposed autoencoders were trained using traditional techniques (e.g. backpropagation algorithm). In addition, the mentioned studies were conducted on older datasets (KDD [17] and NSL-KDD [1]) and a single ML algorithm was evaluated (e.g. SVM, ANN, K-means).

The contributions of the paper can be summarized as follows:

- The autoencoder was trained with differential evolution, using the error of the autoencoder and the accuracy of the ML algorithm (LDA classifier) as an evaluation measure to improve ID performance.
- The evaluation was performed in a more recent ID dataset UNSW-NB15.
- The analysis of the results based on the performance of six different supervised and unsupervised ML algorithms on the encoded data.
- The comparative analysis of the proposed method against PCA.

The main goal of our research is to provide efficient ML-based intrusion detection, but also to improve computational efficiency and preserve data privacy.

This paper has been divided into the following sections. Section 2 explains the details of the proposed method and its main components: autoencoder, differential evolution, and ML algorithms that were used for classification. Section 3 contains the experimental setup. Section 4 presents the results and the discussion of the conducted experiments. Finally, conclusions and plans for future work are given in Section 5.

2 METHOD

The goal of our method is to encode the features using an autoencoder and use the encoded features for intrusion detection with different ML algorithms. The autoencoder is trained with differential evolution, using a combination of autoencoder's error and ML algorithm's accuracy to achieve better performances. A general overview of the method is shown in Figure 1, and all the different parts will be explained in the following subsections.

2.1 Feature Encoding: Autoencoder

An autoencoder [15] is an artificial neural network widely used for dimensionality reduction or feature learning. It maps the input data into a lower-dimensional set of features (encoding phase) and reconstructs the generated set back to the input data (decoding phase). Artificial neural networks are composed of neurons that are grouped into different layers: one input layer, one or more hidden layers, and one output layer. The autoencoder has a special hidden layer, called latent layer, which has fewer nodes than the input layer and forces the network to develop a good representation of the input data [20]. The autoencoder has three main parts:

- Encoder - maps the input into the latent vector.
- Decoder - maps the latent to a reconstructed input.
- Latent layer - contains compressed data (latent vector).

The number of neurons in the input layer and in the output layer is the same and corresponds to the number of features in the dataset. The number of nodes in the latent layer is one of the autoencoder hyperparameters, and it defines how many features the compressed data will have. Neurons are mutually connected using edges, and the strength of each edge is expressed by its weight. The architecture of the autoencoder is shown in Figure 2.

Encoder and decoder mappings, in the simple architecture with one hidden layer, can be described as Eq. (1) and Eq. (2) respectively, where x represents the input data, h the latent vector, σ the sigmoid activation function (Eq. (3)), W/W' the weights, b/b' the biases and x' the reconstructed input.

$$h = \sigma(Wx + b) \tag{1}$$

$$x' = \sigma(W'h + b') \tag{2}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

The similarity between the reconstructed data and the original input data is a performance measure for autoencoders, and the goal of the training process is to reduce the reconstruction error. The autoencoder error can be calculated using the Mean Squared Error (MSE) which measures the average squared difference between the reconstructed input and the actual input (Eq. (4)).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x - x')^2 \tag{4}$$

Weights and biases are initialized at random values and adjusted during training. The autoencoder is an unsupervised learning algorithm, and it can be trained using different training methods. In this paper, we used the differential evolution described in the next subsection.

2.2 Training Autoencoder: Differential Evolution

Differential Evolution (DE) [36, 37] is a population-based optimization algorithm, proposed by Storn and Price in 1995, that belongs to the family of evolutionary algorithms [9]. The population will be composed of many individuals (population size - *PS*), where each one of them will be a combination of weights used in an autoencoder.

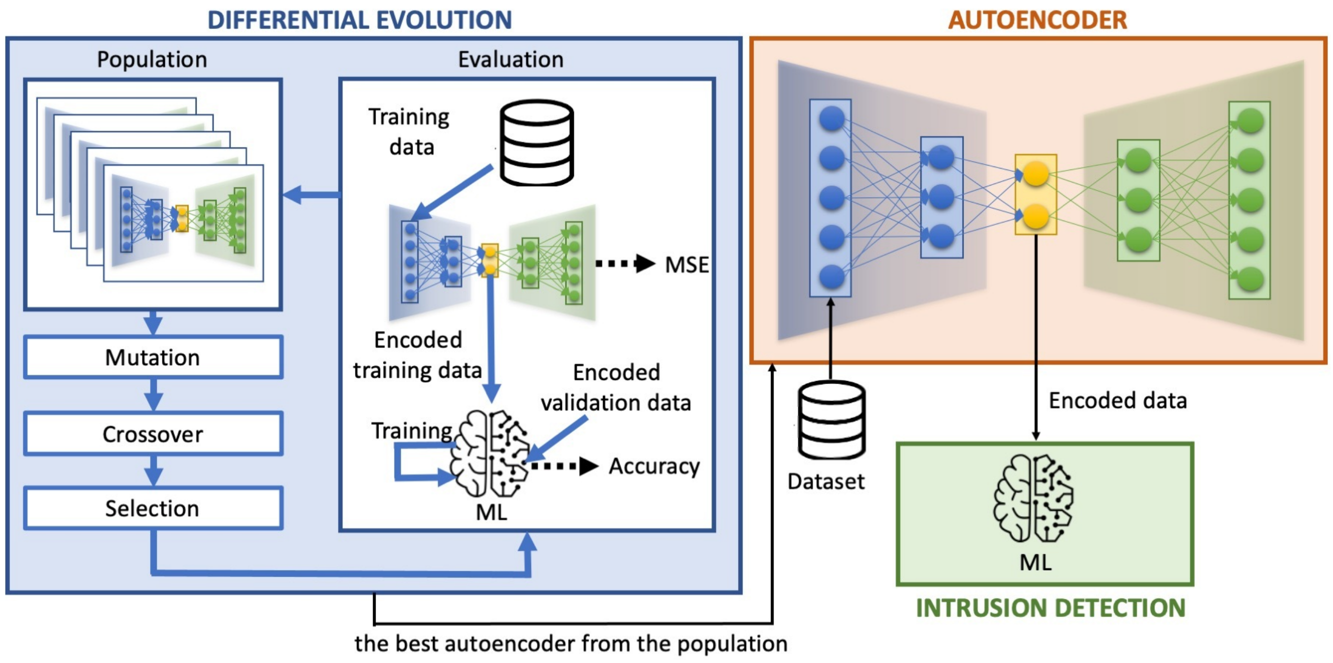


Figure 1: Proposed method

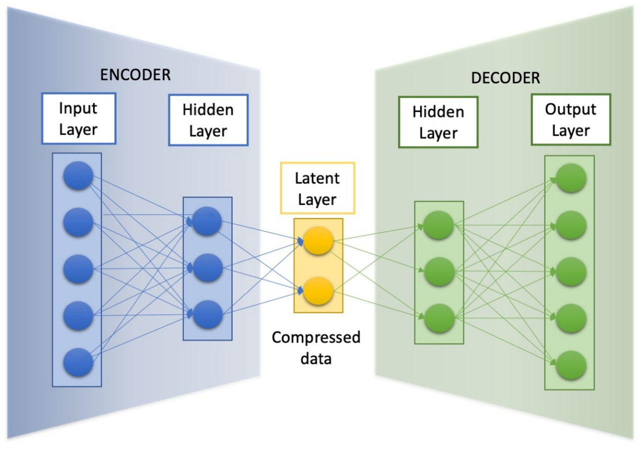


Figure 2: Autoencoder architecture

The i-th individual can be defined as $X_i = \{x_1, \ldots, x_j, \ldots, x_m\}$ where x_j is the weight matrix of the j-th layer in the autoencoder and m is the number of layers. The dimension of the different weight matrices will depend on the number of nodes used in the different layers. For example, a layer with 100 neurons connected to a layer with 10 neurons will have 100×10 as dimension.

At first, all different weights are randomly initialized in the range $[0, 1]$. Then, all individuals will be transformed using three different operators: mutation, crossover, and selection. All these operators together will form a cycle and will be applied a number of times, in other words, for a number of generations. These operators are described as follows.

Mutation: The first step, called mutation, will calculate the difference between different individuals to find a direction to explore a new part of the search space (a new combination of weights). There are many different ways to perform the mutation that were compared in [24] and we selected DE/current-to-best/1 as it obtained the best performance among the tested strategies. DE/current-to-best/1 is described as

$$V_{i,g} = X_{i,g} + F \times (X_{best,g} - X_{i,g} + X_{r_1,g} - X_{r_2,g}) \tag{5}$$

where $V_{i,g}$ represents the i-th mutant vector in generation g, $X_{best,g}$ stands for the best individual in the population, $X_{r_1,g}$ and $X_{r_2,g}$ represent two different randomly selected individuals in the population, and $F \in [0, 2]$ is the mutation factor. The total number of mutant vectors must be equal to PS.

Crossover: After calculating all the different mutant vectors, the second step in DE is to create the same number of offspring as in the population. Every single weight of the i-th offspring (U_i) will be taken either from the i-th individual in the population (X_i) or the i-th mutant vector (V_i) from the current generation (g). This selection will follow

$$U_{i,g}[j][r,c] = \begin{cases} V_{i,g}[j][r,c] & \text{if } rand < CR \\ X_{i,g}[j][r,c] & \text{otherwise} \end{cases} \tag{6}$$

where j stands for the j-th layer in the autoencoder, r and c will give the specific weight inside the layer, $CR \in [0, 1]$ is the crossover rate, and $rand$ is a random value following a uniform distribution within the $[0, 1]$ range.

Selection: The final step in DE is called selection. The fitness values of the i-th individual in the population ($f(X_i)$) and the fitness value of the i-th offspring ($f(U_i)$) are compared, and the best one will survive for the next generation while discarding the other. This process is shown in Eq. (7).

$$X_{i,g+1} = \begin{cases} U_{i,g} & \text{if } f(U_{i,g}) > f(X_{i,g}) \\ X_{i,g} & \text{otherwise} \end{cases} \tag{7}$$

To calculate the fitness values of the individuals and offspring, two metrics have been used: MSE of the autoencoder in the training set and the accuracy of a ML algorithm in the validation set. A linear combination of both values is used as the fitness value, that can be calculated as in Eq. (8).

$$fitness = (1 - MSE) + Accuracy \tag{8}$$

2.3 Classification with Encoded Features: Machine Learning Algorithms

In this paper, we address the binary classification problem (intrusion detection). This section briefly explains the ML algorithms that were used. The first five algorithms are in the SL category, while the last one is in the UL category.

Artificial Neural Network (ANN) [19] are inspired by the human brain. ANNs are composed of artificial neurons that are mutually connected and grouped in different layers. The strengths of connections in ANNs are expressed by weights, and the goal of training process is to reduce the prediction error by adjusting those weights.

Linear Discriminant Analysis (LDA) [11] tries to find a hyperplane that separates the classes with minimized error between the predicted class and the actual class. LDA has a high accuracy for problems with linear separable data.

Support Vector Machine (SVM) [34] also has a goal to find the hyperplane that separates different classes, but the hyperplane is placed with the maximum margin to the support vectors (closer solutions to the hyperplane). SVM has good generalization ability, and it transforms the dimensionality of the data points by using a kernel function, which helps to separate data that are not linearly separable.

Random Forest (RF) [26, 33] groups multiple decision trees to achieve higher precision. Decision tree (DT) is based on rules that are created by evaluating the importance of input parameters from the dataset. Different DTs are created using different parts of the dataset and then combined in RF using boosting ensemble learning methods.

K-Nearest Neighbours (K-NN) [13, 34] is a simple ML algorithm that does not have learning process, it only stores entire training set. For each new example, K-NN calculates the distance from that one to all training examples and selects the majority class from the K nearest neighbors as the predicted class. Different distance measures can be used for KNN, such as the Euclidean distance, Manhattan distance, cosine similarity measure, etc.

K-means [21] is a clustering algorithm that belongs to the category of unsupervised learning. K-means forms K clusters with random centers, assigns the training set examples to the closest cluster and takes the middle point from the assigned points as the new center. This process is repeated until the centers remain unchanged between two iterations. In our experiments, we used K-means for classification. After clusters are formed, each cluster was linked with the predominant class among the cases in that cluster. This means that one class can have more than one cluster.

3 EXPERIMENTAL SETUP

3.1 Network Dataset Description

The experiments presented in this paper were conducted on the well-known network dataset UNSW-NB15 [29] [30], that is widely used in the ID research area. UNSW-NB15 contains a combination of real normal and synthesized attack activities of network traffic. Network traffic was captured in the form of packets, data was preprocessed to create the features, and every entry was labeled either as normal activity or as one of the simulated attacks.

UNSW-NB15 contains nine families of simulated attacks: Fuzzers, Analysis, Backdoors, Denial of Service (DoS), Exploits, Generic, Reconnaissance, Shellcode, and Worms. The dataset has 2,540,044 entries, with an attack distribution of around 13%. In this paper, we use the preconfigured training set, provided in file *UNSW_NB15_training-set.csv*, which has 82,332 records and an attack distribution of around 55%.

The dataset has 49 features classified into six groups: flow features (5), basic features (13), content features (8), time features (9), additional generated features (12), and class labels (2). We used all features except flow features and class labels, resulting in 42 features. There were three different data types: binary, real, and categorical. In the preprocessing phase, we normalized the real values to a range between 0 and 1 and performed one-hot encoding for the categorical values. The total number of features after preprocessing was 190. From class labels, we used the "normal/attack" feature that is provided for binary classification.

3.2 Experimental Settings

All experiments were performed on a HP Zbook with an Intel Core i9-988H CPU @ 2.30GHz and Matlab 2019b was used to implement all algorithms. The dataset was divided into three subsets: training set, validation set, and testing set with a 70%-10%-20% distribution. This procedure was performed three times, which means that all experiments were also repeated three times, and the mean value was used in all comparisons.

The autoencoder is composed of three layers:

(1) Input layer, where its dimension is equal to the total number of features (190).
(2) Latent layer, where its dimension depends on the reduction (Red.) that is selected (1, 2, 5, 10 or 20).
(3) Output layer, where its dimension is equal to the dimension of the input layer (190).

After that, DE is executed using the following parameters:

- Population size (PS): 24
- Mutation factor (F): 0.9
- Crossover rate (CR): 0.9
- Maximum number of generations: 2000

- ML algorithm for fitness: LDA

Finally, the best autoencoder from the population is selected, and different ML methods are tested on the encoded data. The parameters for the different methods are:

- *ANN*: Epochs: 6, Batch size: 256 and Learning rate: 0.01. The layers are:
  - *Input Layer*: Number of neurons: number of features (1, 2, 5, 10, 20 or 190).
  - *Hidden Layer*: Number of neurons: 50, Activation function: Sigmoid.
  - *Output Layer*: Number of neurons: number of classes, Activation function: softmax.
- *SVM*: fitcsvm was used with the "gaussian" kernel function.
- *LDA*: fitcdiscr was used with "DiscimType" equal to "pseudoQuadratic".
- *RF*: fitcensemble was used with predefined parameters.
- *K-NN*: K was set to 5 and the Euclidian distance was used as the distance measure.
- *K-means*: The number of initial centroids (K) was set to 100. The centroid is removed if there are no cases assigned to it.

The decision to select parameters for each algorithm was based on the experiments performed in [35] and [16].

4 RESULTS AND DISCUSSION

4.1 Differential Evolution Convergence

The proposed fitness function ("MSE+Accuracy") was tested and compared to a fitness function that only uses the MSE from the autoencoder ("Only MSE"). The results can be found in Table 1. It can be observed that the MSE is lower as we have a lower reduction, meaning that we are able to better reproduce the original data as the number of encoded features increases. The reason is that more information can be retained. However, the accuracy does not have any correlation with respect to the reduction of features. There is one exception, with Red. 1 the accuracy is 4 percentage points less than with other reductions. This is also observable in Figure 3.

Table 1: Comparison of DE performance with the two different fitness functions. Accuracy is the performance of LDA.

Fitness	MSE + Accuracy		Only MSE
	Accuracy	MSE	MSE
Red. 1	85.1	0.138433	0.025197
Red. 2	89.2	0.104367	0.027733
Red. 5	90	0.073994	0.024107
Red. 10	89.2	0.054438	0.015109
Red. 20	89.7	0.041123	0.010989

If the MSEs of both options of the fitness function are compared with each other, we can observe that it is easier to further decrease the MSE when the accuracy is not used. The reason is that DE with "MSE+Accuracy" uses two different values which sometimes are in conflict with each other. Furthermore, this behavior can be seen in Figure 4. DE with the "Only MSE" fitness function obtains better values faster than the proposed version.

The time expended by DE in order to find the weights of the autoencoder is incremental, ranging from 73.9 minutes (reduction to 1 feature) to 151.8 minutes (reduction to 20 features).

4.2 Performance of Machine Learning Algorithms

The autoencoders obtained by using DE with two different fitness functions ("MSE+Accuracy" and "Only MSE") are used to encode the features and reduce the dimensionality of the problem. A total of six ML algorithms: five SL algorithms (RF, SVM, LDA, ANN and KNN) and one UL algorithm (K-means) have been tested on the encoded and non-encoded data. Different reduction sizes were used for the experiments: 1, 2, 5, 10, and 20 features. The results can be found in Figure 5.

As can be observed, RF is the best ML algorithm on the non-encoded data (No Red. in Figure 5) with 97.43%, followed by SVM (94.41%) and K-NN (92.97%). On the other hand, ANN and LDA obtained the worst results with 72.28% and 74.3%, respectively.

If the encoded data are considered, different interesting findings can be pointed out. The first finding is an obvious difference between the algorithms that had the best performances on the non-encoded data (RF, SVM and KNN) and the others (ANN, LDA and K-means). The first group increases performance as the number of features increases. This is especially visible when the number of features increases from 1 (Red. 1 in Figure 5) to a higher level, with the largest difference for SVM. For the second group, there is no clear trend between the different number of features, including the option of only one feature. Only LDA has a clear difference between using only one feature and the other options. The second finding is that the group that had worse performance on non-encoded data (LDA, ANN and K-means) increased the performance by reducing the number of features. This is especially visible with LDA, where the increased performance is up to +14 percentage points. The reason is that LDA was used to decide on the design of the autoencoder.

On the other hand, if the "only MSE" fitness function is considered, a clear ascending performance can be observed when more features are used.

If we compare the accuracy of ML algorithms that use encoded data from autoencoders obtained by using the two different fitness functions in DE, we can see that "MSE+Accuracy" helped five out of six ML algorithms (SVM, LDS, ANN, KNN and K-means) to perform better for all reduction options. Only in the case of RF accuracy, using "Only MSE" was slightly better for 10 and 20 and worse for the others.

4.3 Execution Time of Machine Learning Algorithms

The execution time for training and testing of six evaluated ML algorithms has been measured and the results can be found in Table 2 and Table 3.

When training time is compared (Table 2), the execution time is always reduced when using encoded data compared to using all features. If the comparison is made with Red.1, the improvement goes from 2.14 times faster (for K-means) to 20 times faster (for RF

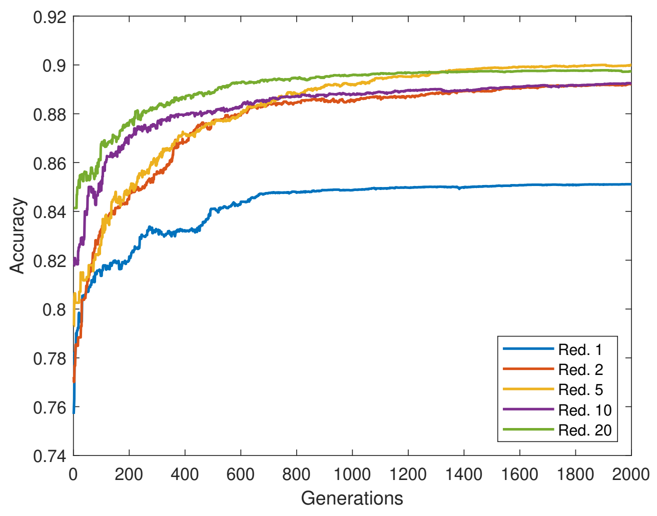


Figure 3: Mean value of accuracy in the validation set by LDA for "MSE+Accuracy" fitness function in DE.

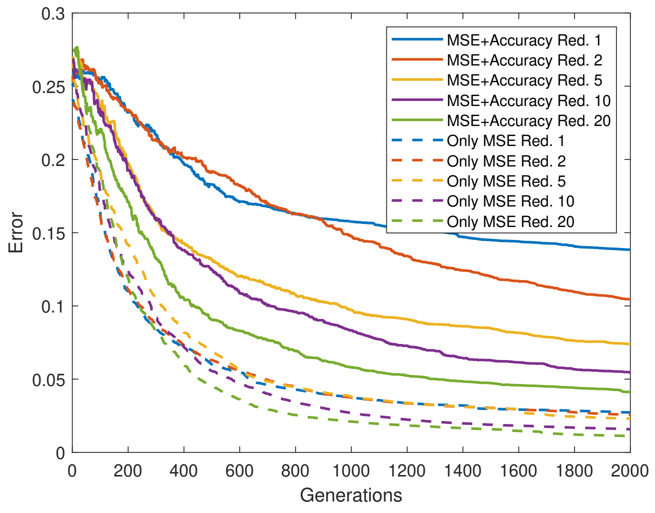
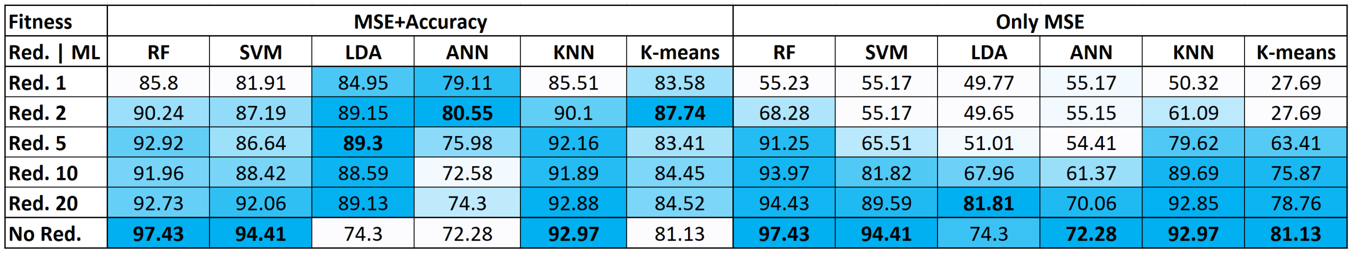


Figure 4: Mean value of the error in autoencoder convergence during the different generations of DE with two different fitness functions "MSE+Accuracy" and "Only MSE".



Fitness	**MSE+Accuracy**						**Only MSE**					
Red. | ML	**RF**	**SVM**	**LDA**	**ANN**	**KNN**	**K-means**	**RF**	**SVM**	**LDA**	**ANN**	**KNN**	**K-means**
Red. 1	85.8	81.91	84.95	79.11	85.51	83.58	55.23	55.17	49.77	55.17	50.32	27.69
Red. 2	90.24	87.19	89.15	**80.55**	90.1	**87.74**	68.28	55.17	49.65	55.15	61.09	27.69
Red. 5	92.92	86.64	**89.3**	75.98	92.16	83.41	91.25	65.51	51.01	54.41	79.62	63.41
Red. 10	91.96	88.42	88.59	72.58	91.89	84.45	93.97	81.82	67.96	61.37	89.69	75.87
Red. 20	92.73	92.06	89.13	74.3	92.88	84.52	94.43	89.59	**81.81**	70.06	92.85	78.76
No Red.	**97.43**	**94.41**	74.3	72.28	**92.97**	81.13	**97.43**	**94.41**	74.3	**72.28**	**92.97**	**81.13**

Figure 5: Heatmap with accuracy of six ML algorithms on the encoded (five different reduction sizes) and non-encoded features for two different DE fitness functions. Darker blue means better performance, while white represent the worse performance per ML algorithm. Values in boldface indicate the best reduction option per ML algorithm.

and LDA). On the other hand, for Red. 20 the improvement range goes from 1.8 times faster (for RF) to 9.5 times faster (for LDA).

When the testing time is considered (Table 3), there is an improvement for all algorithms except RF, which takes the same time independently of the number of features. ANN has the smallest improvement and is already 2 times faster. The biggest difference is achieved for KNN which is 10 times faster with 20 features and 67 times with 1 feature.

4.4 Autoencoders vs Principal Component Analysis

In this subsection, we will compare the proposed method with Principal Component Analysis (PCA) and the results can be found in Table 4.

It can be observed how the proposed method is able to improve the performance of LDA, ANN and K-means for all reduction options. The improvement varies from 7 to 16 percentage points depending on which algorithm is selected. Only for ANN with Red. 20 the results are similar. On the other hand, when RF, SVM and K-means are considered, the results also improve for Red. 1 and Red. 2, while obtaining a similar performance on Red. 5, 10 and 20.

This proves that the proposed method brings a benefit, since if the results are compared with the autoencoder where "only MSE" is considered, PCA outperforms the autoencoder for almost all reduction options.

5 CONCLUSION

This paper presented the feature encoding approach based on autoencoder and differential evolution for the network intrusion detection application. When training the autoencoder with DE, we used an extended fitness function that considers the error of the autoencoder and the accuracy of one ML algorithm during the autoencoder training process. For the experiments presented in this paper, we selected LDA as the ML algorithm to drive the search in DE. The final autoencoders were used in the UNSW-NB15 dataset. The feature set (total of 190 after preprocessing) was encoded into new feature sets with different sizes (1, 2, 5, 10, and 20 features). Six different ML techniques were then evaluated on the encoded feature sets.

Table 2: Training time for six ML algorithms with different number of features (in seconds)

Red. | ML	RF	SVM	LDA	ANN	KNN	K-means
Red. 1	7.21	253.71	0.17	41.8	0	27.15
Red. 2	14.97	240.17	0.17	42.39	0	80.37
Red. 5	16.96	199.46	0.31	41.4	0	22.04
Red. 10	38.61	175.84	0.26	41.57	0	19.31
Red. 20	80.64	162.83	0.37	41.43	0	29.46
No Red.	144.52	593.25	3.53	360.23	0	58.06

Table 3: Testing time for one example for six ML algorithms with different number of features (in miliseconds)

Red. | ML	RF	SVM	LDA	ANN	KNN	K-means
Red. 1	0.114	0.373	0.003	0.055	2.844	0.008
Red. 2	0.117	0.598	0.001	0.055	3.647	0.008
Red. 5	0.086	0.587	0.000	0.053	6.404	0.008
Red. 10	0.111	0.507	0.001	0.053	10.257	0.013
Red. 20	0.117	0.499	0.008	0.053	18.405	0.019
No Red.	0.108	2.657	0.026	0.101	188.437	0.057

Table 4: Results of six ML algorithms after feature encoding with two methods: the proposed method (Autoencoder with DE MSE+Accuracy) and Principal component Analysis. Values in boldface indicate which encoding method is better for a specific reduction option and ML method.

Encoding method	Autoencoder with DE MSE+Accuracy						PCA					
Red. | ML	**RF**	**SVM**	**LDA**	**ANN**	**KNN**	**K-means**	**RF**	**SVM**	**LDA**	**ANN**	**KNN**	**K-means**
Red. 1	**85.8**	**81.91**	**84.95**	**79.11**	**85.51**	**83.58**	79.34	74.59	68.42	68.07	78.23	74.26
Red. 2	**90.24**	**87.19**	**89.15**	**80.55**	**90.1**	**87.74**	84.74	79.18	61.9	68.07	84.15	78.78
Red. 5	**92.92**	**86.64**	**89.3**	**75.98**	**92.16**	**83.41**	91.33	85.52	78.45	68.46	90.82	78.28
Red. 10	91.96	88.42	**88.59**	**72.58**	**91.89**	**84.45**	**92.17**	**89.29**	77.98	69.89	91.38	77.74
Red. 20	92.73	**92.06**	**89.13**	74.3	**92.88**	**84.52**	**92.98**	90.37	80.18	**74.75**	91.74	77.47

The results showed that with much less features, the accuracy was not affected to a high degree. For some of the algorithms, as LDA, ANN and K-means, it is even increasing. Furthermore, reducing the number of features brought a benefit in terms of time consumption (making algorithms up to 20 times faster for training and 67 times faster for testing) and data privacy (since the ML algorithms do not use actual network data). Additionally, we showed that the proposed approach is better for feature reduction than PCA.

As a future work, we would like to use a multi-objective optimization algorithm to consider the trade-off between the different objectives without a linear combination of them, as well as to use another ML method to drive the search. In addition to this, we plan to test the proposed method for attack classification, as well as on different ID datasets.

ACKNOWLEDGMENT

This work has been partially supported by the H2020 ECSEL EU Project Intelligent Secure Trustable Things (InSecTT). InSecTT (www.insectt.eu) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876038. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey.

The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] 2009. NSL-KDD. [https://www.unb.ca/cic/datasets/nsl.html].

[2] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[3] Abdulla Amin Aburomman and Mamun Bin Ibne Reaz. 2017. A novel weighted support vector machines multiclass classifier based on differential evolution for intrusion detection systems. *Information Sciences* 414 (2017), 225–246.

[4] Majjed Al-Qatf, Yu Lasheng, Mohammed Al-Habib, and Kamal Al-Sabahi. 2018. Deep learning approach combining sparse autoencoder with SVM for network intrusion detection. *Ieee Access* 6 (2018), 52843–52856.

[5] Ammar Alazab, Michael Hobbs, Jemal Abawajy, and Moutaz Alazab. 2012. Using feature selection for intrusion detection system. In *2012 international symposium on communications and information technologies (ISCIT)*. IEEE, 296–301.

[6] Mohamed Alloghani, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J Aljaaf. 2020. A systematic review on supervised and unsupervised machine learning algorithms for data science. *Supervised and unsupervised learning for data science* (2020), 3–21.

[7] Md Zahangir Alom and Tarek M Taha. 2017. Network intrusion detection for cyber security using unsupervised deep learning approaches. In *2017 IEEE national aerospace and electronics conference (NAECON)*. IEEE, 63–69.

[8] Megha Aggarwal Amrita. 2013. Performance analysis of different feature selection methods in intrusion detection. (2013).

[9] T Back and H P Schwefel. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation* 1, 1 (1993), 1–23.

[10] M Bahrololum, E Salahi, and M Khaleghi. 2009. Machine learning techniques for feature reduction in intrusion detection systems: a comparison. In *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*. IEEE, 1091–1095.

[11] S Balakrishnama and A Ganapathiraju. 1998. Linear Discriminant Analisys - a brief tutorial. *Institute for signal and information processing* 18 (1998), 1–8.

[12] Anna L Buczak and Erhan Guven. 2015. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials* 18, 2 (2015), 1153–1176.

[13] T.M. Cover and P. Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[14] Fahimeh Farahnakian and Jukka Heikkonen. 2018. A deep auto-encoder based approach for intrusion detection system. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 178–183.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[16] Jesper Hautsalo. 2021. Using Supervised Learning and Data Fusion to Detect Network Attacks. [urn:nbn:se:mdh:diva-54957].

[17] S. Hettich and S. D. Bay. 1999. The UCI KDD Archive. [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science..

[18] Cosimo Ieracitano, Ahsan Adeel, Francesco Carlo Morabito, and Amir Hussain. 2020. A novel statistical analysis and autoencoder driven intelligent intrusion detection approach. *Neurocomputing* 387 (2020), 51–62.

[19] A.K. Jain, J. Mao, and K.M. Mohiuddin. 1996. Artificial neural networks: A tutorial. *Computer* 3 (1996), 31–44.

[20] Mark A Kramer. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal* 37, 2 (1991), 233–243.

[21] Vipin Kumar, Himadri Chauhan, and Dheeraj Panwar. 2013. K-means clustering approach to analyze NSL-KDD intrusion detection dataset. *International Journal of Soft Computing and Engineering (IJSCE) ISSN* (2013), 2231–2307.

[22] Yesi Novaria Kunang, Siti Nurmaini, Deris Stiawan, Ahmad Zarkasi, et al. 2018. Automatic features extraction using autoencoder in intrusion detection system. In *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*. IEEE, 219–224.

[23] Miguel Leon, Tijana Markovic, and Sasikumar Punnekkat. 2022. Comparative Evaluation of Machine Learning Algorithms for Network Intrusion Detection and Attack Classification (in-press). In *2022 international joint conference on neural networks (IJCNN)*. IEEE.

[24] M Leon and N Xiong. 2014. Investigation of Mutation Strategies in Differential Evolution for Solving Global Optimization Problems. In *Artificial Intelligence and Soft Computing*. springer, 372–383.

[25] M Leon, Zenlanter Y., N Xiong, and F Herrera. 2016. Design Optimal Harmonic Filters in Power Systems Using Greedy Adaptive Differential Evolution. In *IEEE 21st International conference on Emerging Technologies and Factory Automation (ETFA)*. 1–7.

[26] Tijana Markovic, Miguel Leon, David Buffoni, and Sasikumar Punnekkat. 2022. Random Forest based on Federated Learning for Intrusion Detection (in-press). In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer.

[27] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).

[28] Tom M Mitchell. 1999. Machine learning and data mining. *Commun. ACM* 42, 11 (1999), 30–36.

[29] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 military communications and information systems conference (MilCIS)*. IEEE, 1–6.

[30] Nour Moustafa and Jill Slay. 2016. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective* 25, 1-3 (2016), 18–31.

[31] Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. 1994. Network intrusion detection. *IEEE network* 8, 3 (1994), 26–41.

[32] Shafigh Parsazad, Ehsan Saboori, and Amin Allahyar. 2012. Fast feature reduction in intrusion detection datasets. In *2012 Proceedings of the 35th International Convention MIPRO*. IEEE, 1023–1029.

[33] Paulo Angelo Alves Resende and André Costa Drummond. 2018. A survey of random forest based methods for intrusion detection systems. *Comput. Surveys* 51, 3 (2018). https://doi.org/10.1145/3178582

[34] S Russell and P Norvig. 2016. Artificial Intelligence: A Modern Approach. In *Malaysia: Pearson Education Limited.*

[35] George Sarossy. 2021. Anomaly detection in Network data with unsupervised learning methods. [urn:nbn:se:mdh:diva-55096].

[36] R Storn and K Price. 1995. *Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces.* Tech Rep. TR-95-012. Comput. Sci. Inst., Berkeley, CA, USA.

[37] Rainer Storn and Kenneth Price. 1997. Differential Evolution –A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359. https://doi.org/10.1023/A:1008202821328

[38] Mahdi Zamani and Mahnush Movahedi. 2013. Machine learning techniques for intrusion detection. *arXiv preprint arXiv:1312.2177* (2013).