

Evolving neural networks for threat process detection using autoencoder to build a fitness function.¹

Henry D. Navarro H.^{a,*,**}, Héctor Bullejos^b and Carmelo Garrido^b and Elena Naranjo^b

^a *Research and Development Lab, Vision Analytics, Avenida de Europa 19, 28224, Pozuelo de Alarcón, Madrid, Spain*

E-mail: contact@henrynavarro.org

^b *Research and Development Labs, Capgemini Engineering, Calle Campezo, 1, 28022, Madrid, Spain.*

E-mail: engineering@capgemini.com

Abstract. Threat detection is one of the main focus of several studies in the cybersecurity area and it has become one of the main focuses of researchers. Threat may occur in any forms like viruses or phishing attack with different purposes. Once the intruder has filtered into our system, the primary step is to try to detect the malicious process to prevent it from causing damage to our system. In this paper, we present a hybrid method using deep neural networks to obtain a fitness function and evolutionary algorithms to detect and kill malicious processes on a simulated network system in streaming.

Keywords: Neuroevolution, Genetic Algorithms, Threat detection, Autoencoders

1. Introduction

Cybersecurity has become into one of the main topics of the last years due to...

In order to simulate a network, attack computers with a malicious process and having an environment under control, we have created a virtual network with 20 virtual instances using open source softwares. We start from the fact that the malicious process has infiltrated in our system either using ransomware, social engineering or any other common technique in modern cybersecurity. We collect the system data of these virtual instances using open source libraries. This data will be used to train neural networks, and detect and kill the malicious process using evolutionary algo-

rithms.

Evolve neural networks has been used in different tasks but mainly in train neural networks to play video games [?], [?], [?], although there are other applications as music generation [?] and modelling biological phenomena [?]. These neural network training methods rely on genetic algorithm operations to obtain the best agent to perform a specific task. However, there is no too much research related with this topic and cybersecurity as we will see in the section 2.

Any evolutionary algorithm needs a fitness function in order to determine what agents or individuals are the best in each generation. In our case, the best agents will be those who “kill” the malicious process in each generation which will result in instances which take their “normal states”. In order to characterize this normal state, we use an ingenious solution for this need, using an artificial neural network called autoencoder. This neural network will learn the healthy state of each

¹Footnote in title.

*Corresponding author. E-mail: contact@henrynavarro.org.

**Do not use capitals for the author’s surname.

instance and using the mean square error (MSE) between the inputs and outputs of the neural networks, we will know how infected is the instance, so the more close this value to zero the more healthy is the state of the instance, so we will use the better agents following this criteria. Then, using the simple command of any Linux system has, we kill the process who is negatively affecting our virtual instance.

2. Related work

Autoencoders are neural networks that consist of an encoder to generate a vector of features in the latent space (of smaller dimension) and from the input data and a decoder, which seeks to reconstruct the input data from this latent vector [?]. This type of neural network is used in anomaly detection [?], [?], natural language processing [?] and dimensionality reduction [?], [?].

Others researchers have proved that autoencoders can be used to detect attacks as Denial of Services [?]. In their paper, [?], they use an autoencoder to classify different denial of services attacks. However, our objective is not only to classify the state of a virtual machine between two states: healthy and under attack but, to get a measure to quantify how infected an virtual instance is. For this, we train this neural network just with the healthy state data and we will use the mean square error (MSE) as a fitness function to choose the best agents in threat detection, the more close this value to zero the more healthy is the state of the instance.

As for the topic of deep neuroevolution, there is no direct application in the area of cybersecurity. However, other researchers have applied genetic algorithms to improve network security [?]. In most of these papers, a binary gene is reached out in order to block or detect certain networks attacks as Syn-Flood, Smurf and others, for this purpose, common operations in genetic algorithms as selection, crossover, mutation are used.

Different investigations in malware detection have been carried out using deep neural networks for them [?], however, these researchers train their models using algorithms that involve gradients. In this paper, we present other way to train a neural network that can detect a malicious process that create idle tasks that af-

fects common system values like CPU, memory RAM, hard drive, etc. This process is detected and killed in order to return the healthy state in a virtual instance.

3. Methodology

3.1. Network simulation

The virtual network to emulate the attacks was built and simulated based on the fact that an intruder had already obtained access to the network and already represents an internal threat, that is, the file that executes the malicious process is already found in the virtual instances. In the figure 1 we saw an architecture of our virtualized network where we have 20 instances using an a network orchestrator with 4 physical machines.

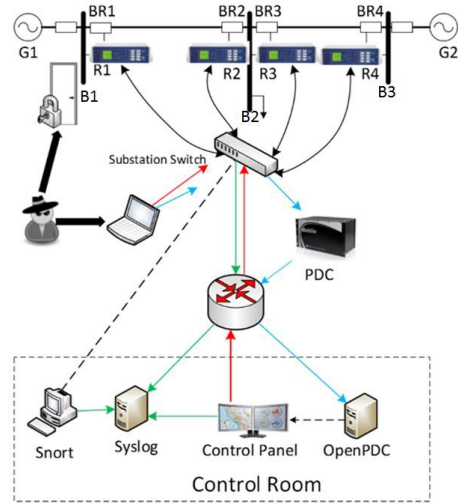


Fig. 1. ESTA NO ES LA IMAGEN, AQUÍ TIENE QUE IR UNA IMAGEN DE LA RED, TENGO QUE PEDIRLE AYUDA A FERNANDO A VER SI TIENE TIEMPO.

These instances send their system data to a NoSQL database where we monitorize the variables that we explain in the next section and these are the features we use to develop our models.

3.2. Dataset

The dataset has been generated first extracting the normal traffic of a virtualized environment to get the “healthy” data that a system could have, to get this normal behaviour we used a software tool able to extract 97 features related mainly with cpu, memory ram or

usage of hard drive that we split into 10 main groups. Then, the two cyber attacks have been sent to this virtualized network and extracting the data to obtain two new classes: A malicious process developed by us that we will call *Logic Bomb*, which is a process that affects the regular parameters like CPU, memory, disk creating idle processes by this task.

3.2.1. Central Processing Unit (CPU) features

They refer to the load characteristics that the CPU of the instance may have. A malicious process acting on the instances is expected to drive CPU usage across the entire instance in order to crash it.

3.2.2. Core features

These features are similar to CPU variables, with the difference that they are disaggregated by core. Due to how virtualized instances have been created (1GB of memory, 1 core only), these variables are expected to have a high correlation with those of the CPU.

3.2.3. Disk Input/Output (Disk I/O) features

Disk I / O operations include both read and write or Input / Output (usually defined in KB / s) involving a physical disk. In simple words, it is the speed with which the data transfer takes place between the hard disk drive and RAM, or basically it measures the input / output time of the active disk. It is a performance measure and is therefore used to characterize storage disks as HDD, SSD, and SAN. A malicious process is expected to constantly perform read and write operations to cause a saturation of this hardware.

3.2.4. Entropy

Entropy available on the system.

3.2.5. Filesystem

This set of features refer to file system statistics on disk. Some malicious processes create an infinite loop that creates files indefinitely in order to saturate it.

3.2.6. Memory Swap

When the physical memory or RAM in our system is full, we proceed to use the *Memory Swap* in our systems. In this process, the inactive pages of our memory are moved to the swap space, creating more memory resources. This space is especially useful when a system does not have RAM; however, the swap space is located on the hard drive and is therefore slower to access. Therefore it should not be considered as an alternative to RAM. As indicated above, a malicious

process is expected to crash RAM memory and from there, begin to consume the resources of swap memory.

3.2.7. Memory hugepages

Hugepages are useful in managing virtual memory on Linux systems. As the name implies, they help manage large pages in memory that are larger than the default (usually 4KB). Hugepages is useful for both 32-bit and 64-bit configurations. Hugepages sizes can range from 2MB to 256MB, depending on the kernel version and hardware architecture. A malicious process is expected to increase the values of this variable.

3.2.8. Socket summary

These variables refer to the summary of open socket metrics in the system. A socket is nothing more than a communication channel between two programs that run on different computers or even on the same computer. Malicious processes seeking to attack the network are expected to affect these variables.

4. Anomaly detection classification for fitness function

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, configuring the target values to be equal to the inputs. That is, the response variable that the machine learning algorithm tries to learn is such that $y^{(i)} = x^{(i)}$, where $x^{(i)} = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$ are the input variables detailed in section 3.2 and the appendix ??.

5. Anomaly detection classification for fitness function

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, configuring the target values to be equal to the inputs. That is, the response variable that the machine learning algorithm tries to learn is such that $y^{(i)} = x^{(i)}$, where $x^{(i)} = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$ are the input variables detailed in section 3.2 and the appendix ??.

5.1. Autoencoder architecture

An autoencoder is made up of two parts, an encoder that we will denote as f_ϕ and a decoder that we denote with g_θ . An autoencoder neural network will be denoted as $h_{W,b} = g_\theta \circ f_\phi$, where W and b are the weight matrix and the vector “bias” or bias of the neural network and h is the final transformation function or hypothesis.

We will denote n_l as the number of layers in our network. In our case, for the autoencoder we have taken a network with three hidden layers, therefore for our case $n_l = 5$. We denote the l -th layer as L_l , therefore L_1 is the input layer and L_{n_l} is the denoting the output layer. We also denote the parameters $(W, b) = (W^{(i)}, b^{(i)})$, with $i = 1, \dots, 5$ and where $W^{(l)}$ are the weight matrices (weights) of each layer and $b^{(i)}$ the bias vector (bias) associated with the connections between the unit (neuron) j in layer l , and unit i in layer $l + 1$. Likewise, we denote $f_{.l}$ the activation function that “connects” layer $l - 1$ with the layer l . And finally, we denote s_i as the number of neurons (units) of the layer i .

Due to the high dimensionality of the dataset that we have used, a selection of variables is made, which we explain in the results section 7.2.1. However, we have selected the number of neurons in such a way that in the encoder part $s_i = \lfloor \frac{s_{i-1}}{2} \rfloor$, while for the part of the decoder $s_i = \lceil 2 \times s_{i-1} \rceil$, where $\lfloor \cdot \rfloor$ represents the integer part of a real number.

Like any neural network, it requires activation functions that allow connecting all the units (neurons) of each layer. For our autoencoder, we have selected the activation functions as follows:

$$f_{.2} = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

$$f_{.3} = ReLU(x) = x^+ = \max(0, x),$$

$$f_{.4} = f_{.2} = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

$$f_{.5} = f_{.3} = ReLU(x) = x^+ = \max(0, x)$$

We will write $a_i^{(l)}$ to denote the activation (the value at the output) of the unit i in the layer l . For $l = 1$, we

use $a_i^{(1)} = x_i$ to denote the i -th input. That is to say,

$$z^{(l)} = W^{(l-1)}a^{(l-1)} + b^{(l-1)}$$

$$a^{(l)} = f_{.l}(z^{(l)})$$

Graphically, the architecture of the autoencoder can be seen in figure 2.

The autoencoder is trained with the well-known algorithm called stochastic gradient descent with 60 epochs. From there, the objective is to determine from the mean square error of the input data with the output data a decision threshold for which we will say that the virtual instance is in a healthy state, in this way, we show how we can use the autoencoder rebuild error to classify rare events (anomalies). If the rebuild error is “high”, we will classify it as an instance failure, this is done with the algorithm 1.

Algorithm 1 Anomaly detection algorithm with the autoencoder

Input: Validation dataset $X_{val} = \{x^{(1)}, \dots, x^{(n)}\}$, autoencoder $h_{W,b} = g_\theta \circ f_\phi$.

Train dataset with normal behaviour X , Anomaly train data $x^{(i)}$ $i = 1, \dots, N$, threshold α

```

1:  $\phi, \theta \leftarrow$  Train the autoencoder with normal data  $X$ 
2: for  $i=1$  to  $N$  do
3:   error de reconstrucción( $i$ ) =  $\|x^{(i)} - g_\theta(f_\phi(x^{(i)}))\|_2^2$   $\triangleright$  Mean Square Error (MSE)
4:   if error de reconstrucción( $i$ )  $> \alpha$  then
5:      $x^{(i)}$  is an anomaly
6:     pred $\leftarrow$ 1
7:   else
8:      $x^{(i)}$  is not an anomaly
9:     pred $\leftarrow$ 0
10:  end if
11: end for
```

Output: Vector of classifications.

On the other hand, to achieve our objective, we will have to determine what is that threshold from which we will say that there is an anomaly. To find this decision threshold, we seek to test for all reconstruction errors in our training data (healthy) and determine which of them is the optimal one that maximizes the accuracy in the confusion matrix, as shown in the algorithm. 2.

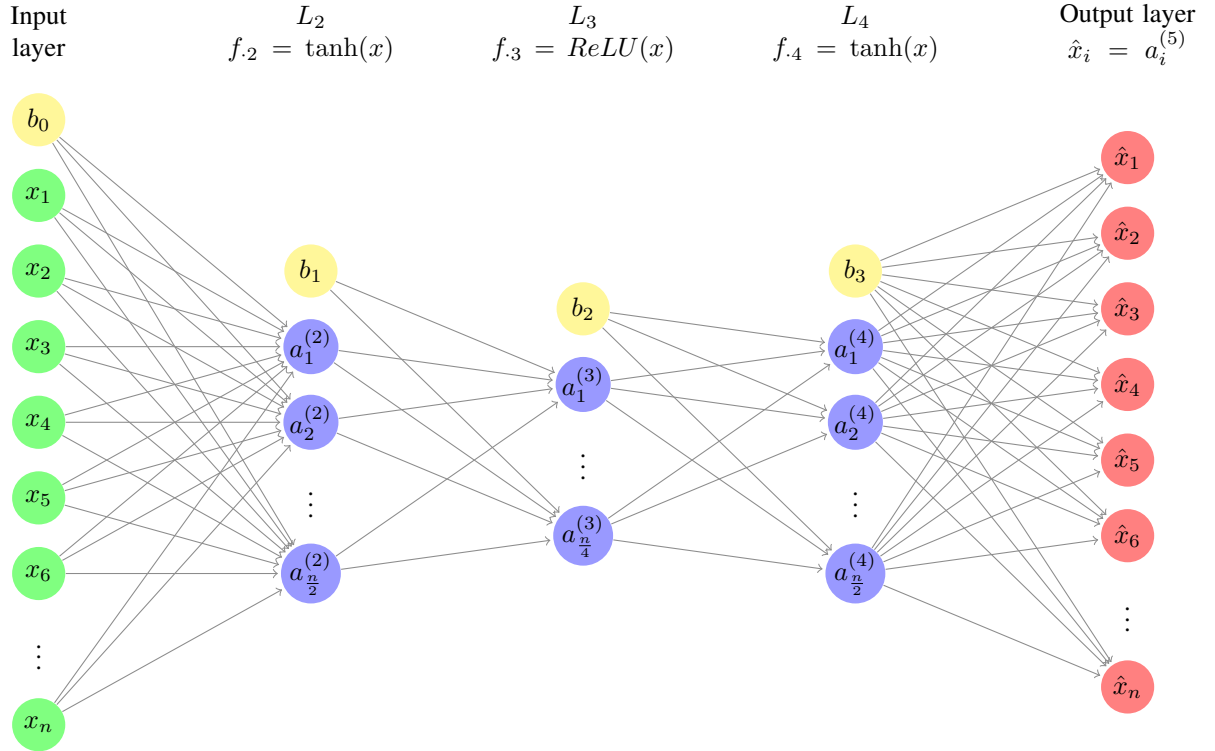


Fig. 2. Architecture of the neural network of the autoencoder type used for anomaly detection

6. Evolutionary algorithm approach

6.1. Features for agents

To characterize those malicious processes in an instance, we study some features that may anticipate this result. These variables to be studied do not directly describe the instance, but rather the processes that are running in it. Remember that in our case, we do not have a dataset of processes classified as malicious, but the hybrid algorithm of machine learning and genetic algorithms, must be able to detect and mitigate it, using the correct order just as any algorithm would do. reinforcement learning, with the difference that this time it does not learn using policy gradients. In our case, we have separated the variables into four groups.

6.1.1. File descriptors

The file descriptor is a non-negative integer that uniquely identifies the files opened in a session. Each process is allowed to have up to nine file descriptors open at one time. The bash shell reserves the first three file descriptors (0, 1, and 2) for special purposes [?].

6.1.2. Central Processing Unit (CPU)

They refer to the load characteristics that the CPU may have for each process. A malicious process acting on the instances is expected to drive up CPU usage in order to crash it.

6.1.3. Memory

This set of variables refer to the RAM memory consumption that each process performs. A malicious process should carry out a high consumption of this type of resources.

6.1.4. Temporal features

As the name indicates, in this set are the variables that are related to time.

6.2. Architecture of the agents (neural networks) of the population

Following the notation in the section 5.1, our neural network has parameters $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$. That is, in our case, we have $W^{(1)} \in \mathbb{R}^{15 \times 8}$, while $W^{(2)} \in \mathbb{R}^{1 \times 15}$. For our case, we use the softmax func-

Algorithm 2 Obtaining the optimal α threshold for anomaly detection

Input: Train dataset $X_{train} = \{X_{train}^{(1)}, \dots, X_{train}^{(n)}\}$, Validation dataset $X_{val} = \{X_{val}^{(1)}, \dots, X_{val}^{(n)}\}$, autoencoder $h_{W,b}$.

```

 $\hat{X}_{train} \leftarrow h_{W,b}(X_{train})$     ▷ Evaluate the training
data in the autoencoder.
 $Max\_MSE\_train \leftarrow$ 
 $\max \left( \sum_{i=1}^n \left( X_{train}^{(i)} - \hat{X}_{train}^{(i)} \right)^2 \right)$     ▷ From the
above list we choose the highest root mean square
error.
 $\hat{X}_{val} \leftarrow h_{W,b}(X_{val})$     ▷ Same for validation data.
 $MSE_{val} \leftarrow \sum_{i=1}^n \left( X_{val}^{(i)} - \hat{X}_{val}^{(i)} \right)^2$ 
 $partition \leftarrow Max\_MSE\_train / 1000$ 
Inicializa  $\alpha$ ,  $Accuracy_{optimal}$ ,  $MSE_{optimal}$  con
valores cero.
    ▷ 1000 threshold partitions are tested, from 0 to
 $Max\_MSE\_train$  and the best is saved:
for  $i=1$  to 1000 do
     $y_{val}^{(i)} \leftarrow anomaly(MSE_{val}, \alpha)$     ▷ Aplicar el
algoritmo 1, vector de 1's y 0's
     $Accuracy_{current} \leftarrow$  Obtener accuracy a partir
de  $y_{val}, y_{val}^{(i)}$ 
    if  $Accuracy_{current} > Accuracy_{optimal}$  then
         $Accuracy_{optimal} \leftarrow Accuracy_{current}$ 
         $\alpha \leftarrow MSE_{current}$ 
    end if
     $MSE_{current} \leftarrow MSE_{current} + partition$ 
end for

```

Output: Optimal threshold α .

tion as a network hypothesis. That is, we will hypothesize $h : \mathbb{R}^2 \rightarrow [0, 1]^2$ as the softmax function of the last layer of each neural network:

$$h_{W,b}(x) = \left(\frac{e^{a_1^{(3)}}}{\sum_{j=1}^2 e^{a_j^{(3)}}}, \frac{e^{a_2^{(3)}}}{\sum_{j=1}^2 e^{a_j^{(3)}}} \right) \quad (1)$$

Therefore, for the logical bomb attack, $n^{[2]} = 2$ because what we will obtain will be a probability vector as follows:

$$(P_{no}, P_{kill}) = (P(\text{no kill process} | X = x_1, \dots, x_n), P(\text{kill process} | X = x_1, \dots, x_n))$$

such that $P_{no} + P_{kill} = 1$, where x_i are the monitored variables for that process, defined in section 6.1. So the activation function in the last hidden layer will be given by the function softmax 1.

In our case, the neural network that we have chosen can be seen in the diagram we can see in figure 2.

6.3. Evolving Neural Networks

In our case, each individual/agent is a neural network known as a multilayer perceptron (MLP) with $n^{[0]}$ input units (neurons in the input layer). In our case $n^{[0]} = 8$, this due to the variables of the processes to analyze (See section 6.1), $n^{[1]} = 15$ hidden units (neurons in the hidden layer) this with the intention that in the hidden layer we have twice as many neurons as in the input layer, and $n^{[2]} = 2$ output units (neurons in the output layer).

6.4. Initialization

We will have for each generation, λ neural networks (initial population size), where λ will be the maximum number of instances that can be virtualized in our environment according to computer resources. The initialization of the weights $W_{ij}^{[l]}$ of the λ neural networks is done using the general rule to establish the weights in a neural network, which is to establish them in such a way that they are close from scratch without being too small, for this, we define the function HEWEIGHTS as follows:

Algorithm 3 He Initialization

```

1: function HEWEIGHTS(agent)
2:    $n \leftarrow$  number of nodes of layer  $l$     ▷ agent is the
neural network
3:    $y \leftarrow \frac{1.0}{\sqrt{n}}$ 
4:   agent.weights  $\leftarrow U(-y, y)$     ▷ random
numbers with distribution  $U(-y, y)$ 
5:   return agent
6: end function

```

Therefore, the initialization operation of the deep neuroevolution algorithm is a configuration of λ MLP's that will have an initialization of the weights $W_{ij}^{[l]}$ with uniform distribution as seen in the algorithm 4:

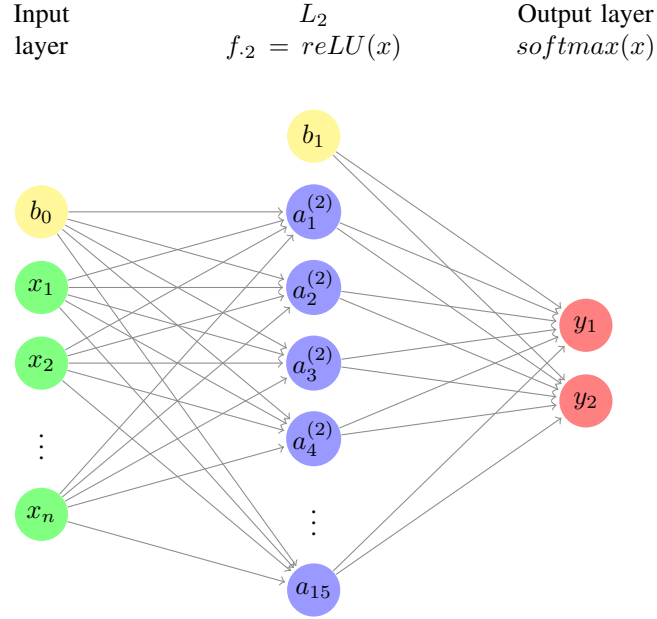


Fig. 3. Neural network architecture used for attack mitigation.

Algorithm 4 Initialization

Input: population size or number of agents λ , number of neurons in input each layer: $n^{[0]}, n^{[1]}, n^{[2]}$, weights initialization function ϕ .

- 1: agents \leftarrow empty list
- 2: **for** $i = 1, \dots, \lambda$ **do**
- 3: init_agent \leftarrow create a MLP with $n^{[0]}, n^{[1]}, n^{[2]}$ neurons in each layer and ReLU as activation function.
- 4: init_agent \leftarrow HEWEIGHTS(init_agent) \triangleright function that return weights using He's initialization.
- 5: agents[i] \leftarrow init_agent \triangleright add to list the agent initialized
- 6: **end for**

Output: λ neural networks with He's initialization.

6.4.1. Evaluation

This operation is the one that indicates how good are the agents that result from the initialization or after each operation (selection, crossing, mutation, elitism). For this, we have adjusted a machine learning model trained only with healthy values (without infecting the virtual instance) of the machine. This is where the autoencoder steps in to provide a measure through mean square error, which we have detailed in Section 5.

The objective of this model is that it be able to provide a measure that allows determining an anomaly in the data of the virtual instances. For this we have based on the textit mean square error (MSE) between the input data x_1, x_2, x_3, \dots of the autoencoder (instance data) and the data at the output of the autoencoder $\hat{x}_1, \hat{x}_2, \hat{x}_3, \dots$.

Therefore, the evaluation operator or *fitness function* of the Deep Neuroevolution algorithm will be given by:

$$fitness = MSE = \frac{1}{n^{[0]}} \sum_{i=1}^{n^{[0]}} (x_i - \hat{x}_i)^2 \quad (2)$$

where x_1, x_2, x_3, \dots are the input data of the autoencoder (instance data) and $\hat{x}_1, \hat{x}_2, \hat{x}_3, \dots$ is the data in output the output of the autoencoder. **The closer the fitness function is to zero, the better the agent will be.**

6.4.2. Selection and Crossing

After the evaluation, those individuals/agents (from the n neural networks) with the best fitness (2) should be selected, in our case, a fitness closer to zero will be an indicator of better agent.

Now, within the operations that we will define are two of the most important: Selection and Crossing. For the first operation, the name is quite intuitive and what is sought is to select the best agents according to the fitness function (the closer to zero, the better). For our case, we select the best n_{top} where

$$n_{top} = \left\lfloor \frac{-1 + \sqrt{1 + 4 \cdot 2 \cdot \lambda}}{2} \right\rfloor + 1 \quad (3)$$

and $\lfloor x \rfloor$ is the *floor* function. This is because the crossover operation (which we explain later) needs to reproduce $\lambda - 1$ agents from the best ones, which requires a number n_{top} such that $\sum_{i=1}^{n_{top}} i = \lambda$. Therefore, the equation 3 is nothing more than the solution of $\frac{n_{top}(n_{top}+1)}{2} = \lambda$, where λ is the number of agents in the initial population P_0 .

The crossing operation we have defined from the best n_{top} agents, it is intended from these to create a population of size $\lambda - 1$ crossing the weights W_{ij} of an agent “father” and several “mother” agents with a probability of 0.5. This is nothing more than a Bernoulli trial where a coin is tossed, if a result comes out, the son will have the weight W_{ij} of the father, otherwise he will keep the weight W_{ij} of the mother.

6.4.3. Mutation

In evolutionary algorithms it is convenient that for the new descendants formed by selection and crossing, some of their genes (in our case the weights of the units or neurons) can be subjected to a mutation with a low random probability of a change. This implies that some weights of the neurons of the initial λ neural networks, will be modified with a “small” change that depends on a value known as *mutation power* [?] to which we will call σ .

The mutation occurs to maintain diversity within the population, without this operation, the values of the weights W_{ij} of the neural networks (agents) would only maintain the values obtained from initialization.

That is, we are adding to the weights W_{ij} of the units (neurons) of each neural network, a value $\sigma \cdot \mathcal{N}(0, 1)$ with $\sigma = 0.02$. For properties of the normal distribution, we will have to simply add a value with distribution $\mathcal{N}(0, 0.02^2) = \mathcal{N}(0, 0.0004)$. As you can see, this very small value will prevent the weights from having extreme changes but at the same time it will allow these weights to vary beyond the values they have

Algorithm 5 Selection and Crossover

```

1:  $n_{top} = \left\lfloor \frac{-1 + \sqrt{1 + 4 \cdot 2 \cdot \lambda}}{2} \right\rfloor + 1$ 
2: function SELECTION AND Crossover( $agents, n_{top}$ )
3:    $agents\_top =$  select best  $n_{top}$  agents from the list agents.
4:    $childrens =$  empty list
5:    $max\_id = 1$   $\triangleright$  Initialize a loop end parameter
6:   while  $max\_id \leq n-1$  do
7:     for  $i = 1, \dots, max\_id$  do
8:        $mother = agents\_top[i]$   $\triangleright$  Neural network mother
9:        $father = agents\_top[max\_id]$   $\triangleright$  Neural Network father
10:      for each unit of neural network  $W_{ij}^{[parent]}$  do
11:         $coin = U(0, 1)$   $\triangleright$  Get random number with uniform distribution
12:        if  $coin \leq 0.5$  then
13:           $W_{ij}^{[children]} = W_{ij}^{[father]}$   $\triangleright$  Update each weight of each unit
14:        else
15:           $W_{ij}^{[children]} = W_{ij}^{[mother]}$   $\triangleright$  Update each weight of each unit
16:        end for  $childrens[i] = new\_children$ 
17:      end for
18:    end while
19:    Return  $childrens$ 
20: end function

```

taken in the initialization operation (Algorithm 4). This value σ could be considered as the analogous value to the learning rate in the case of gradient descent. We have therefore defined the following algorithm:

6.4.4. Elitism

Elitism in our case refers to a selection of the neural network that is best able to mitigate an attack. For this, from the n_{top} top agents that are selected (3), what is known as *tournament elitism* is performed. That is, we tested a total of m times each agent (neural network) in the virtualized environment with the infected instances and the average of the *fitness* 2 is calculated for these m times. This average will determine which is the best after the m executions. Finally, the elite agent (neural network) is added to the list of child agents (*mutated_childrens*), thus keeping us the best of all **unmutated** and without any alteration, allowing us to keep the one with the best mitigation characteristics of at-

Algorithm 6 Mutation

```

1:  $\sigma = 0.02$  ▷ mutation power
2: mutated_childrens= empty list
3: function MUTATION(childrens) ▷
   As input it needs the list returned by selection and
   crossover operation (Algorithm 5)
4:   for  $i = 1, \dots, \text{total of childrens}$  do
5:     for each weight  $W_{ij}$  of unit of neural net-
       work do
6:        $r_{norm} = \text{get random number } \mathcal{N}(0, 1)$ 
7:        $W_{ij} = W_{ij} + \sigma \cdot r_{norm}$  ▷ Update each
       weight of each unit
8:       new_children=assign to this neural
       network the weights  $W_{ij}$ .
9:     end for
10:    mutated_childrens[i]=new_children
11:  end for
12:  Return mutated_childrens
13: end function

```

tacks.

Algorithm 7 Elitism

```

1:  $n_{top} = \left\lceil \frac{-1 + \sqrt{1 + 4 \cdot 2 \cdot \lambda}}{2} \right\rceil + 1$ 
2: function ELITISM(mutated_childrens, agents,  $n_{top}$ )
3:   agents_top= select best  $n_{top}$  agents from the
   list agents.
4:   for  $i = 1, \dots, n_{top}$  do
5:     agent_test = agents[i]
6:     fitness_agent= empty list
7:      $j=1$ 
8:     repeat
9:       test the agent in the virtualized envi-
       ronment
10:      fitness_agent[j]=calculate the fitness
       using
11:      the autoencoder 2.
12:       $j=j+1$ 
13:    until  $j=m$ 
14:  end for
15:  fitness_mean[i]= $\frac{1}{m} \sum_{j=1}^m \text{fitness\_agent}[j]$ 
16:  elite=select agent with best fitness_mean
17:  new_generation=add elite to mu-
   tated_childrens
18:  Return new_generation
19: end function

```

On the other hand, we want to penalize when the neural network decides to kill all the processes of the

instance, this is because if we do not do this, the neural network will learn to reduce the MSE given by the autoencoder based on kill all the processes of the instance, so when the agent applies `kill` to all processes, the command will be changed to not kill any processes. Finally the scheme of the neuroevolution algorithm can be seen in Figure 4.

7. Experiments and Results

In this section, we present the results of the experiments carried out following the proposed algorithms in the previous sections.

7.1. Feature selection

Due to the high dimensionality of the dataset, it is necessary to discard certain features in order to improve the performance of the model. Firstly, to reduce the number of variables, a previous study of each variable was made, comparing the results between the different states of the machine. The first variables to delete were those are not affected in either state of the virtual instance. In this way, 26 of the 97 initial variables that only accounted for computational load when training the neural network were eliminated, in addition to overtraining.

It is well known that multicollinearity in data considerably reduces the predictive power of many machine learning models. Despite the fact that an autoencoder is not a linear model, it is convenient to reduce this high dimensionality of the data in any case and an important measure to do so, is through correlation.

In this way, it was first established that variables were strongly related to each other (a correlation coefficient higher than 0.7) and once the relationships were obtained, a study was carried out on the variables themselves, that is, the definition of the variables themselves. Thus, if two variables explained the same information, one of them would be discarded. It was also tried to have at least one representative variable for each module in order to have more complete information on the system. After this study, a total of 58 out of 97 variables were discarded.

7.2. Results

The results show that it is possible to determine the state of a instance using autoencoders and also detect

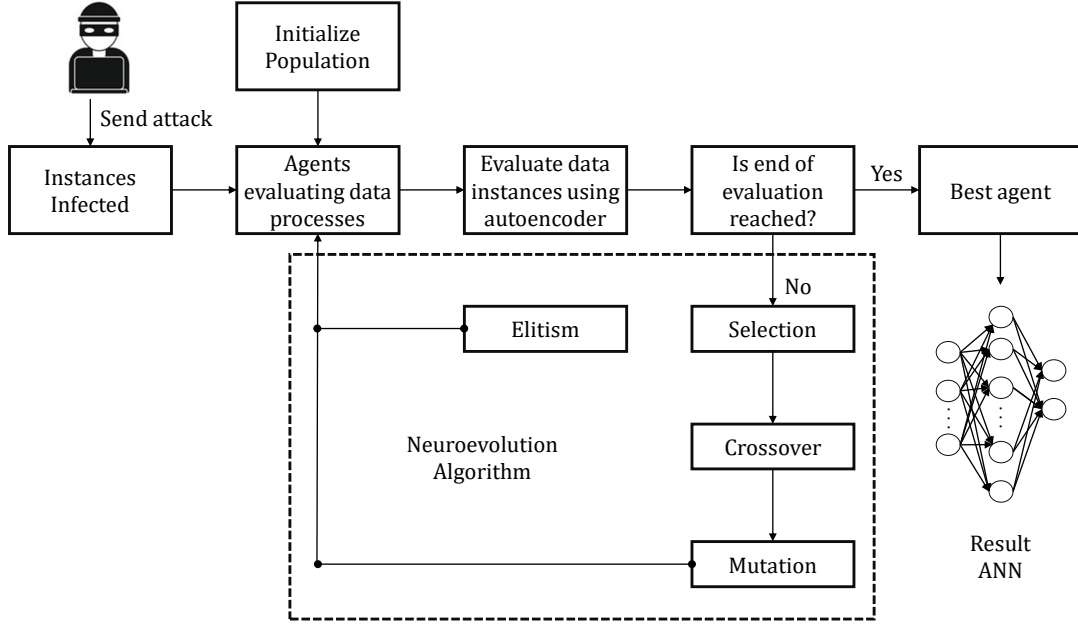


Fig. 4. Neuroevolution algorithm flow.

malicious processes that affect the normal behaviour of a computer. So the results are presented in two steps: anomaly detection results and evolutionary algorithms for neural networks.

7.2.1. Autoencoder model results

The autoencoder model was trained with only the healthy state of the data using the stochastic gradient descent algorithm with 60 epochs. The mean square error was monitored during the training process, the results are shown in Figure 5.

After training the model, it is necessary to determine the decision threshold from which the state of the machine is considered an anomaly, for this, the Algorithm 2 was used, giving a threshold $\alpha \approx 1.1945 \dots$, the closer value to this the state of the instance is, the greater the consideration will be as a healthy state of the instance. Therefore the agents of the evolutionary algorithm should eliminate the process that is affecting the instance and will be selected as the best “parents” those that after killing the process, lead to the state of the instance at values close to this α value. The results are shown in Figure 6.

With this decision threshold, a confusion matrix is obtained in order to try using the autoencoder also as a

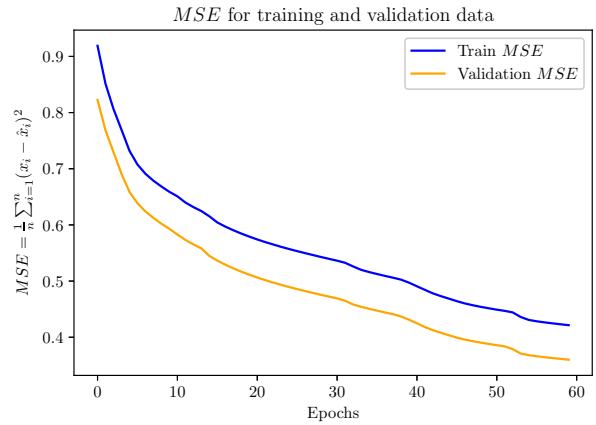


Fig. 5. N° epochs vs. MSE . Mean square error for a trained (Train) and validated (Validation) data for the autoencoder neural network with data from healthy machines after feature selection.

classifier. The results are shown in Table 2.

Then, a 10-fold cross validation is done and finally we get the following performance measures (mean of the 10-fold results), which usually are taken into account to determine the goodness of fit.

Variable 1	Variable 2	Correlación
load 15	load norm 15	1.0
load norm 1	load 1	1.0
memory used bytes	memory free	1.0
memory actual used bytes	memory actual free	1.0
load norm 5	load 5	1.0
filesystem free	filesystem available	0.99
memory used bytes	memory used pct	0.991
memory free	memory used pct	0.99
memory actual free	memory actual used pct	0.99
memory actual used bytes	memory actual used pct	0.99
entropy available bits	entropy pct	0.99
core idle pct	cpu idle pct	0.99
core system pct	cpu system pct	0.99
core user pct	cpu user pct	0.99
core nice pct	cpu nice pct	0.99
cpu iowait pct	core iowait pct	0.99
diskio io time	diskio write count	0.99
process summary sleeping	process summary total	0.98
diskio write bytes	diskio io time	0.98
cpu total pct	core idle pct	0.98
cpu total pct	cpu idle pct	0.98
fsstat total size used	fsstat total size free	0.987
diskio io time	diskio write time	0.98
fsstat total size used	filesystem used bytes	0.97
diskio write bytes	diskio write count	0.96
filesystem used bytes	filesystem used pct	0.96
fsstat total size free	filesystem used pct	0.96
diskio write count	diskio write time	0.95
fsstat total size free	filesystem used bytes	0.94
diskio iostat write request per sec	diskio iostat busy	0.94
filesystem free	filesystem free files	0.92
filesystem available	filesystem free files	0.92
diskio iostat await	diskio iostat write await	0.92
socket summary all count	socket summary udp all count	0.91
socket summary tcp all established	socket summary tcp all count	0.917
diskio iostat queue avg size	diskio iostat busy	0.90

Table 1

Pearson correlation (ρ) table for variables with correlation coefficient $\rho > 0.90$.

		Real	
		L. Bomb	Healthy
Predicted	L. Bomb	1629	174
	Healthy	135	5457

Table 2

Confusion matrix build from threshold decision α . The classification is done following a MSE on infected instance.

These results show that our autoencoder model can classify states of the virtual machines using anomaly detection and being trained just with one kind of data (healthy state).

In this section we will present the most important results of this paper. We will see how a neural network trained with a hybrid machine learning algorithm with evolutionary algorithms is able to reduce the effect of

$$Accuracy = \frac{\text{Correctly classified}}{\text{Total of samples}} = 0.9632$$

$$Precision = \frac{\text{Correctly classified as infected}}{\text{Samples predicted as infected}} = 0.9335$$

$$Recall = \frac{\text{Correctly classified as infected}}{\text{Samples actually are infected}} = 0.9136$$

7.2.2. Neural evolution results

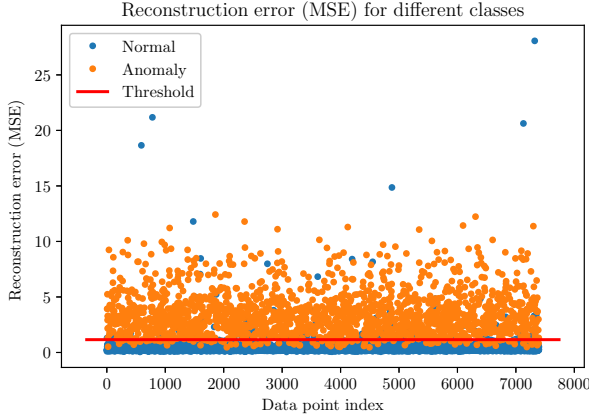


Fig. 6. Threshold decision from the mean square error (MSE). The red line represents the threshold decision $\alpha \approx 1.1945 \dots$ to classify the anomaly.

these attacks on virtual instances.

From the generation 10, it is appreciated (Figure 7) that the best agent is able to reduce the mean squared error at the output of the autoencoder. For this, the best agent kills the processes that are considered by it as malicious, so the objective is not only kill all processes but kill the specific process who is affecting the virtual instance, so as we can see in Figure 7 the best agent reduce the number of processes that it pretends to kill until finally kill just the malicious process.

8. Conclusions

In this paper we have shown other way to reduce the effect of cyber attacks and as well as another application that has the neuroevolution algorithms who has never applied in cyber security. We have confirmed the results of other researchers [?] using an autoencoder to classify cyber attacks, although in our case we have used other methodology and other cyber attack. The autoencoders can give you a measure of how close to a healthy state a virtual instance is, this allow us to combine this measure with a neuroevolution algorithm to select the best agents, giving as a result an ingenious hybrid algorithm able to detect malicious processes that are being running in our shell.

Appendix

A. Appendix: Description of each variable

A.1. Central Processing Unit (CPU) Features

cpu_cores: The number of CPU cores present on the host. The non-normalized percentages will have a maximum value of $100\% \cdot \text{cores}$. The normalized percentages already take this value into account and have a maximum value of 100%.

cpu_user_pct: The percentage of CPU time spent in user space. On multi-core systems, you can have percentages that are greater than 100%. For example, if 3 cores are at 60% use, then the `system.cpu.user.pct` will be 180%.

cpu_system_pct: The percentage of CPU time spent in kernel space.

cpu_nice_pct: The percentage of CPU time spent on low-priority processes.

cpu_idle_pct: The percentage of CPU time spent idle.

cpu_iowait_pct: The percentage of CPU time spent in wait (on disk).

cpu_irq_pct: The percentage of CPU time spent servicing and handling hardware interrupts.

cpu_softirq_pct: The percentage of CPU time spent servicing and handling software interrupts.

cpu_steal_pct: The percentage of CPU time spent in involuntary wait by the virtual CPU while the hypervisor was servicing another processor.

cpu_total_pct: The percentage of CPU time spent in states other than Idle and IOWait.

A.2. Core features

core_user_pct: The percentage of CPU time spent in user space.

core_system_pct: The percentage of CPU time spent in kernel space.

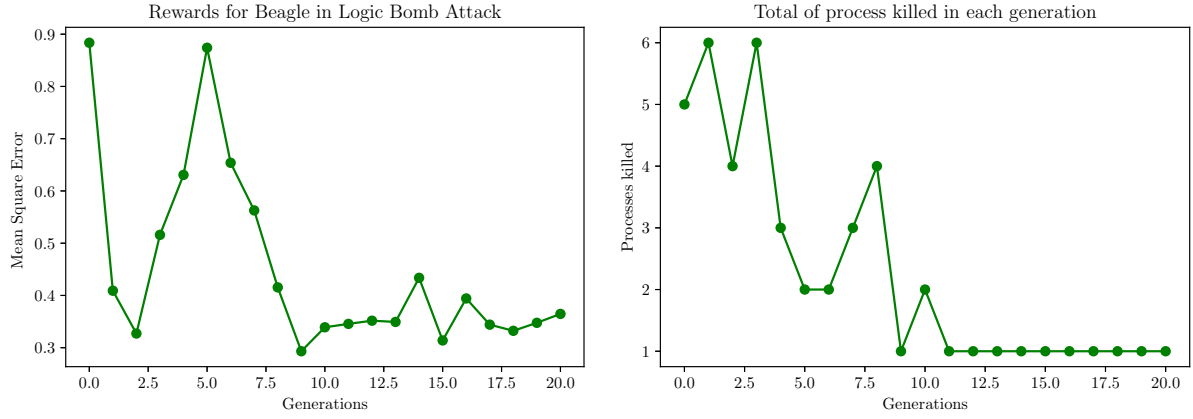


Fig. 7. (Left) Rewards (MSE) for the best agent in each generation using an algorithm to evolve neural networks and (right) total processes that are killed in each generation evolving neural networks using as fitness function the MSE given by the autoencoder (Equation 2).

core_nice_pct: The percentage of CPU time spent on low-priority processes.

core_idle_pct: The percentage of CPU time spent idle..

core_iowait_pct: The percentage of CPU time spent in wait (on disk).

core_irq_pct: The percentage of CPU time spent servicing and handling hardware interrupts.

core_softirq_pct: The percentage of CPU time spent servicing and handling software interrupts.

core_steal_pct: The percentage of CPU time spent in involuntary wait by the virtual CPU while the hyper-visor was servicing another processor. Available only on Unix.

A.3. Disk Input/Output (Disk I/O) features

diskio_read_count: The total number of reads completed successfully.

diskio_write_count: The total number of writes completed successfully.

diskio_read_bytes: The total number of bytes read successfully. On Linux this is the number of sectors read multiplied by an assumed sector size of 512.

diskio_write_bytes: The total number of bytes written successfully. On Linux this is the number of

sectors written multiplied by an assumed sector size of 512.

diskio_read_time: The total number of milliseconds spent by all reads.

diskio_write_time: The total number of milliseconds spent by all writes.

diskio_io_time: The total number of milliseconds spent doing I/Os.

diskio_iostat_read_request_merges_per_sec: The number of read requests merged per second that were queued to the device.

diskio_iostat_write_request_merges_per_sec: The number of write requests merged per second that were queued to the device.

diskio_iostat_read_request_per_sec: The number of read requests that were issued to the device per second.

diskio_iostat_wirte_request_per_sec: The number of write requests that were issued to the device per second.

diskio_iostat_read_per_sec_bytes: The number of Bytes read from the device per second.

diskio_iostat_read_await: The average time spent for read requests issued to the device to be served.

diskio_iostat_write_per_sec_bytes: The number of Bytes write from the device per second.

diskio_iostat_write_await: The average time spent for write requests issued to the device to be served.

diskio_iostat_request_avg_size: The average size (in bytes) of the requests that were issued to the device.

diskio_iostat_queue_avg_size: The average queue length of the requests that were issued to the device.

diskio_iostat_await: The average time spent for requests issued to the device to be served.

diskio_iostat_service_time: The average service time (in milliseconds) for I/O requests that were issued to the device.

diskio_iostat_busy: Percentage of CPU time during which I/O requests were issued to the device (bandwidth utilization for the device). Device saturation occurs when this value is close to 100%.

A.4. Entropy

entropy_available_bits: The available bits of entropy.

entropy_pct: The percentage of available entropy, relative to the pool size of 4096.

A.5. Filesystem

filesystem_available: The disk space available to an unprivileged user in bytes.

filesystem_files: The total number of file nodes in the file system.

filesystem_free: The disk space available in bytes.

filesystem_free_files: The number of free file nodes in the file system.

filesystem_total: The total disk space in bytes.

filesystem_used_bytes: The used disk space in bytes.

filesystem_used_pct: The percentage of used disk space.

fsstat_count: Number of file systems found.

fsstat_total_files: Total number of files.

fsstat_total_size_free: Total free space.

fsstat_total_size_used: Total used space.

fsstat_total_size_total: Total space (used plus free).

A.6. Memory Swap

memory_swap_pct: Total swap memory.

memory_swap_used_bytes: Used swap memory in bytes.

memory_swap_free: Available swap memory.

memory_swap_used_pct: The percentage of used swap memory.

A.7. Memory hugepages

memory_hugepages_total: Number of huge pages in the pool.

memory_hugepages_used_bytes: Memory used in allocated huge pages in bytes.

memory_hugepages_used_pct: Percentage of huge pages used.

memory_hugepages_free: Number of available huge pages in the pool.

memory_hugepages_reserved: Number of reserved but not allocated huge pages in the pool.

memory_hugepages_surplus: Number of over-committed huge pages.

memory_hugepages_default_size: Default size for huge pages.

A.8. Socket summary

socket_summary_all_count: All open connections.

socket_summary_all_listening: All listening ports.

socket_summary_tcp_memory: Memory used by *Transmission Control Protocol* (TCP) sockets in bytes, based on number of allocated pages and system page size.

socket_summary_tcp_all_orphan: A count of all orphaned tcp sockets.

socket_summary_tcp_all_count: All open TCP connections.

socket_summary_tcp_all_listening: All TCP listening ports.

socket_summary_tcp_all_established: Number of established TCP connections.

socket_summary_tcp_all_close_wait: Number of TCP connections in *close_wait* state.

socket_summary_tcp_all_time_wait: Number of TCP connections in *time_wait* state.

socket_summary_udp_memory: Memory used by UDP sockets in bytes, based on number of allocated pages and system page size.

socket_summary_udp_all_count: All open UDP connections.

B. Appendix: Description of each variable for the agents.

B.0.1. File descriptors

process_fd_limit_hard: limit hard on the number of file descriptors opened by the process. The hard limit can only be increased by root.

process_fd_limit_soft: limit soft on the number of file descriptors opened by the process. The process can change the limit soft at any time.

process_fd_open: the number of file descriptors opened by the process.

B.0.2. Central Processing Unit (CPU)

process_cpu_total_norm_pct: Percentage of CPU time that the process consumes since the last event. This value is normalized by the number of CPU cores and ranges from 0% to 100%.

B.0.3. Memory

process_memory_size: The total in bytes of virtual memory that the process has.

process_memory_rss_bytes: The Resident Set Size (RSS) in bytes. The proportion of memory used by a process that is held in main memory (RAM), that is, the memory that the process occupied in main memory or RAM.

process_memory_share: the shared memory in bytes that the process uses.

B.0.4. Temporal features

process_cpu_start_time_seconds: The time (in seconds) since the process started.

B.1. Body of the text

- The use of first persons (i.e., “I”, “we”, “their”, possessives, etc.) should be avoided, and can preferably be expressed by the passive voice or other ways. This also applies to the Abstract.
- A research paper should be structured in terms of four parts, each of which may comprise of multiple sections:
 - * Part One is problem description/definition, and a literature review upon the state of the art
 - * Part Two is methodological formulation and/or theoretical development (fundamentals, principle and/or approach, etc.)
 - * Part Three is prototyping, case study or experiment;
 - * Part Four is critical evaluation against related works, and the conclusion.
- A survey paper may skip Part Three, but should multiply Part Two and elaborate Parts One and Four.

- An application paper may lightly touch Part Two but should elaborate Part Three, with Parts One and Four similar to the a research paper.

In any article it is unnecessary to have an arrangement statement at the beginning (or end) of every (sub-) section. Rather, a single overall arrangement statement about the whole paper can be made at the end of the Introduction section.

C. Typographical style and layout

C.1. Type area

The `iosart2c.cls` document class has been designed to produce the right layout from your \LaTeX input. Authors are requested to strictly follow these instructions. *The provided class file iosart2c must not be changed.*

The text output area is automatically set within an area 16.2 cm horizontally and 21.5 cm vertically. Please do not use any \LaTeX or \TeX commands that affect the layout or formatting of your document (i.e. commands like `\textheight`, `\textwidth`, etc.).

C.2. Font

The font type for running text (body text) is 10 point Times New Roman. There is no need to code normal type (roman text). For literal text, please use `typewriter` (`\texttt{}`) or `sans serif` (`\textsf{}`). *Italic* (`\emph{}`) or **boldface** (`\textbf{}`) should be used for emphasis.

C.3. General layout

Use single (1.0) line spacing throughout the document. For the main body of the paper use the commands of the standard \LaTeX “article” class. You can add packages or declare new \LaTeX functions if and only if there is no conflict between your packages and the `iosart2c.cls`.

Always give a `\label` where possible and use `\ref` for cross-referencing.

Class option “onecolumn” should be used only if it is necessary for the layout of your article:
`\documentclass[onecolumn]{iosart2c}`.

Use class option “crrready” in order to remove the page numbers from your article.

C.4. Title page

Use sentence case for the title.

Use `\thanks{}` command for footnotes in `\title`, `\author` and `\address` commands.

Do not use capitals for author’s surname. Add “and” before the last author. Do not add a period after the last keyword.

C.5. (Sub-)section headings

Use the standard \LaTeX commands for headings: `\section`, `\subsection`, `\subsubsection`, `\paragraph`. Headings will be automatically numbered.

C.6. Footnotes and endnotes

Footnotes should only be used if absolutely essential. In most cases it is possible to incorporate the information in the text. If used, they should be kept as short as possible. The footnotes are numbered automatically. If the footnotes take up more space than roughly 10% of the type area, list them as endnotes, before the References. Footnotes within the text should be coded with the command `\footnote{Text}`, endnotes with the command `\endnote{Text}`. The command `\theendnotes` prints out the endnotes. The additional package `endnotes` is required for the endnotes (`\usepackage{endnotes}`). If you do not have it in your \LaTeX environment, see your system manager, or download them from <http://www.ctan.org>.

C.7. References

References should be collected at the end of your paper (environment `thebibliography`). References should be listed alphabetically in the style presented in the section **References** at the end of these instructions. Use the command `\cite` to refer to the entries in the bibliography so that your accumulated list corresponds to the citations made in the text body.

C.8. Figures

C.8.1. General remarks on figures

The text should include references to all figures. Refer to figures in the text as Figure 1, Figure 2, etc., **not** with the section number included, e.g. Figure 2.3, etc. Do not use the words “below” or “above” when referring to the figures.

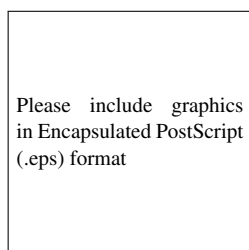


Fig. 8. Figure caption.

Do not collect figures at the back of your article, but incorporate them in the text.

Position figures at the top or bottom of a page, near the paragraph in which the figure is first mentioned. Figures should not have text wrapped alongside.

Each figure should have a self-explanatory caption. Place the figure caption *below* the figure.

All figures coded with `figure` and `\caption` will be numbered automatically.

On maps and other figures where a scale is needed, use bar scales rather than numerical ones of the type 1:10,000.

C.8.2. *Quality of illustrations*

Use only Type I fonts for the lettering in illustrations. Do *not* use illustrations taken from the Internet. The resolution of images intended for viewing on a screen is not sufficient for the printed version of the journal. If you are incorporating screen captures, keep in mind that the text may not be legible after reproduction (using a screen capture tool, instead of the Print Screen option of PC's, might help to improve the quality). Preferably the images should be enclosed as PostScript files – best as EPS data.

- Line art should have a minimum resolution of 600 dpi;
- grayscales (incl photos) should have a minimum resolution of 300 dpi (no lettering), or 500 dpi (when there is lettering);
- do not save figures as JPEG, this format may lose information in the process;
- do not use figures taken from the Internet, the resolution will be too low for printing;
- do not use color in your figures if they are to be printed in black & white, as this will reduce the print quality (note that in software the default is often color, so you should change the settings);
- for figures that should be printed in color, please send a CMYK encoded EPS.

C.8.3. *Color figures*

It is possible to have figures printed in color, provided the cost of their reproduction is paid for by the author. Please contact editorial@iospress.nl for a quotation if you wish to have figures printed in color. See section ?? for the required file formats.

C.9. *Tables*

The text should include references to all tables. Refer to tables in the text as Table 1, Table 2, etc., not with the section number included, e.g. Table 2.3, etc. Do not use the words “below” or “above” referring to the tables.

Position tables at the top or bottom of a page, near the paragraph in which the table is first mentioned. Tables should not have text wrapped alongside.

Code your tables using \LaTeX environments `table` and `tabular`. Each table should have a brief and self-explanatory caption that should be put above the table. Do not use the period at the end of the table caption. If the table does not fit into one column it may be placed across both columns using `\begin{table*}` so that it appears at the top of a page.

All tables coded with `table` and `\caption` will be numbered automatically.

Column headings should be brief, but sufficiently explanatory. Standard abbreviations of units of measurement should be added between parentheses. Vertical lines should not be used to separate columns. Leave some extra space between the columns instead. Any explanations essential to the understanding of the table should be given in footnotes at the bottom of the table. SI units should be used, i.e., the units based on the metre, kilogramme, second, etc.

Tables should be presented in the form shown in Table 3. Their layout should be consistent throughout.

C.10. *Equations*

Do not put in equation numbers, since this is taken care of automatically. The equation numbers are always consecutive and are printed in parentheses flush with the right-hand margin of the text and level with the last line of the equation. For multi-line equations, use the `eqnarray` environment. Refer to equations in the text as Eq. (1), Eqs. (3) and (5).

Table 3
Table caption

Dataset	Models	α_1	α_2	α_3	α_4	α_5	α_6
CSDS	Linear	0.164	0.22	0.123	0.3	0.200	0.258
	Logistic	0.189	0.155	0.157	0.201	0.154	0.144
KCDS	Linear	0.155	0.183	0.160	0.218	0.176	0.156
	Logistic	0.187	0.125	0.151	0.184	0.187	0.125

D. Fine tuning

D.1. Type area

Check once more that all the text and illustrations are inside the type area and that the type area is used to the maximum.

D.2. Capitalization

Use sentence case in the title and the headings.

D.3. Page numbers and running headlines

You do not need to include page numbers or running headlines. These elements will be added by the publisher.

E. Submitting the paper

Submit the following to the journal Editorial office, online submission form or Editor-in-Chief (whichever is applicable for the journal):

1. The main \LaTeX document as well as other required files (eps, etc.).
2. Identical high resolution PDF file with all fonts embedded. First produce a Postscript file from \LaTeX with DVIPS version 5.56 or higher. The option "-M" (don't make fonts) should be indicated. Use Adobe Acrobat Distiller and choose the job option Press-Optimized.) Please make sure you do not submit more than one version of any item.

F. Copyright

Authors submitting a manuscript do so in the understanding that they have read and agreed to the terms of the IOS Press Author Copyright Agreement posted in the 'Authors Corner' on www.iospress.nl.

F.1. Quoting from other publications

An author, when quoting from someone else's work or when considering reproducing figures or tables from a book or journal article, should make sure that he/she is not infringing a copyright. Although in general an author may quote from other published works, he should obtain permission from the holder of the copyright if he wishes to make substantial extracts or to reproduce tables, plates or other figures. If the copyright holder is not the author of the quoted or reproduced material, it is recommended that the permission of the author should also be sought. Material in unpublished letters and manuscripts is also protected and must not be published unless permission has been obtained. Submission of a paper will be interpreted as a statement that the author has obtained all the necessary permission. A suitable acknowledgement of any borrowed material must always be made.

G. How to order reprints, a pdf file, journals, or IOS Press books

The corresponding author of a contribution to the journal is entitled to receive 1 copy of the journal free of charge, unless otherwise stated. Free copies will not be provided for conference proceedings and abstract issues. An order form for reprints, additional journal copies or a pdf file will be provided along with the pdf proof. If you wish to order reprints of an earlier published article, please contact the publisher for a quotation at IOS Press, Fax: +31 20 6870039. E-mail: editorial@iospress.nl. An author is entitled to 25% discount on IOS Press books.

References