A Comparative Investigation of Neuroevolutionary methods in Neural Network-based Intrusion Detection System

Sanyat Hoque and Ken Ferens
Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada
{hoques@myumanitoba.ca , ken.ferens@umanitoba.ca}

Abstract—Machine learning algorithms include but not limited to neural network, genetic algorithm, support vector machine techniques and other classifiers that have been extensively used in the intrusion detection systems. However, the detection performance of these algorithms with the security data set is not satisfactory due to over/under-fitting problems. In this work, the authors have proposed a comparison of dynamically evolving artificial algorithm to not only optimize the synaptic weights and error of the neural network learning but also optimize the number of layers and number of neurons per layer based on the input data set. More specifically, in this work, the authors would like to analyze the improvements of detection performance using data dependent neural network structure and test the performance over a real time labeled wireless data set. The analysis shows that the performance has improved and the resultant neural network is able to converge more optimally and quickly to the steady state minimum values with a modular and symmetric neural network.

Keywords—Neuroevolution, Intrusion Detection System (IDS), Lindenmayer System, Genetic algorithm, Artificial neural network, Dynamic trees Introduction

I. INTRODUCTION

The main goal of this paper is to build an intrusion detection system [1] for Wireless Local Area Network based on WLAN MAC layer frame. Three machine learning algorithms were used in order to compare and contrast the accuracy and false positives on a public dataset called Aegean Wi-Fi Intrusion Dataset (AWID). In this paper, the first machine learning technique used for comparison is a biologically inspired neuroevolution technique with synaptic neuron constraint using Lindenmayer System [2] with memory that helps to implement modularity and hierarchy, ie. recursive use of same substructure and modular decomposition of problem used in making an intrusion detection. The second algorithm used is a direct encoded heuristic based neuroevolution technique [3] with connection constraints where heuristic based mutation and crossover operators of node addition, deletion and crossover of nodes among best individuals highly optimize a network. In the above two mentioned techniques, the introduction of penalty factors in using neurons and input features allows a more efficient and compressed network structure that reduces search space dimension. To obtain faster convergence and optimization, the heuristic based Neuroevolution (NE) technique and Lindenmayer System with memory neuroevolution algorithm optimize connection weights, network shapes, as well as input, features simultaneously via the introduction of penalty factors to the overall fitness of each individual of a population of each artificial neural network. For comparison with the previous two direct and indirect encoded Neuroevolutionary techniques with synapse constraints, another technique is developed with fixed topology (FT) based direct encoding scheme but without any connection constraint. Therefore, this algorithm is not expected to show any form of structural symmetry or modularity. All three techniques follow an evolutionary process that selects only the fittest individual out of a set of the population with different topologies. The population topology starts from very simple network structure with one hidden layer and grows into a complex network until convergence.

Connection constraint-based neuroevolution techniques are first implemented in a XOR dataset for testing and then implemented on the publicly available Aegean Wi-Fi Intrusion Dataset. In the experiment, it is expected to show properties of modularity, regularity or symmetric nature of a network [4] improves convergence rate via penalties in synapse connections [5] within the network structure. Henceforth, a specific methodology such as penalty factors in neuron connections is needed to implement such complex properties in a neural network. Adding connection constraints within a network infrastructure to an artificial neural network within its hidden nodes and input nodes [3] [5] introduces these properties very nicely. To investigate these properties in developing an intrusion detection system, these techniques are first tested in XOR dataset and then implemented in AWID dataset to compare and contrast.

In evolutionary based artificial neural network, encoding information refers to the transfer of mapping from genotype to phenotype. Traditionally, evolutionary algorithms use direct encoding process where every element is mapped directly to phenotype [3]. In contrast to indirect encoding, elements in genotype are mapped indirectly [2] to its phenotype allowing the recursive use of the same substructure, for example, a single element in genotype describes at least two structures in its phenotype. Compact structures of a genotype lead to a much more complex and regular structure at a molecular level through reuse of elementary genomic information [4]. This recursive use helps in the evolution of modular, regular and hierarchical phenotypes. This further reduces dimension search space and allows faster convergence in solution space. Regularity is the compressibility of information in a dataset which can have symmetric or modular repetition. Therefore, indirect encoding usually would outperform direct encoding with and without connection constraints in a more regular or symmetric data set.

The rest of this paper is organized as follows: Section II presents a literature survey on existing neuroevolution models. A more detailed analysis about each methods is provided in Section 3. The corresponding datasets of XOR dataset and AWID datasets are discussed in Section 4. The simulation results are discussed in Section 5. Section 6 concludes the paper and provides a scope of future work.

II. RELATED WORK

In [6], several machine learning techniques were used to build a wireless intrusion detection system with improved feature reduction as well as using a variety of machine learning techniques such as naïve bayesian to compare the best classifier for intrusion detection. Authors in [5] extensively discussed connection constraint methods proving faster convergence is possible within a highly specific structured modular and regular network. This brings about the hierarchy which is the recursive composition of different organized structures of network nodes. In [4], authors achieved modularity via the hierarchical use of same structure through the process of production rules and penalizing a number of neuron connections in an entire neural network including input connections. Research in [7], shows modularity makes clusters of nodes with dense connectivity to form a dense cluster with some nodes but loosely connected to other nodes outside the cluster.

One of the key features in reaching faster learning through increased level of evolution of a Neuroevolution technique is to introduce modularity within the architecture of the network. In modular networks, every individual of a population rearranges its molecular nodal structures to reduce its dimension search space. Modularity is the localization of function within a larger space. A common misconception is that regularity entails modularity which is not true. For example, a single wheel in a unicycle is a module whereas to four wheels on a car are a regular repetition of a wheel module, i.e. same substructure. Therefore, adding connection cost to each synapse of a network architecture also introduces modularity and regularity able to take advantage of a symmetrical dataset.

III. NEUROEVOLUTION ALGORITHMS

In this paper, an investigation is analyzed via comparing two Neuroevolution algorithms with direct encoding and indirect encoding scheme coupled with synapse connection constraint and with another direct encoding based neuroevolution algorithm without synapse penalty. For the sake of comparison, this fixed topology based Neuroevolution algorithm without connection constraint is used in this paper to be used in XOR dataset and AWID dataset in conjunction with others. In a direct encoding scheme, every connections and node are specified in the genome and absence of penalty means unnecessary connections would be present that does not improve scalability. Results indicate that first two techniques showed promising results while direct encoding without connection cost underperforms in comparison with the other Neuroevolutionary techniques.

A. *Fixed Topology Neuroevolution and Dropout Mechanism*

In the FT based Neuroevolutionary algorithm created in this paper, only one individual in each niche is made that only undergoes mutation but no node addition or deletion. It would only increase its topology size from one generation to the next transferring its chromosomes from generation I to i+1. To counter the problem of over-fitting as it becomes a complex structure with deep layers, a dropout technique [8] is introduced to avoid overfitting and improve generalization. Since only one individual exists within its niche, the individuals do not go through the crossover. Each individual passes its genome from one generation the next starting from very simple neural network architecture to become a very complex non-modular structure until it evolves to a certain threshold fitness. In the previous two techniques that are based on connection constraints, allowing the population to compete within its own class of topologies protects innovations as well as helps to create competition between a highly topologically diversified population.

1) *Stochastic training with droupouts*

In [8], hidden neurons among layers are removed randomly with probability 0.5 for each training example iteration in Fig. 1. The removal process is done by setting hidden units to zero before passing it to itssigmoid activation function. Since all input connections to the particular neuron are set to zero. There would be no back propagation via gradient descent through that neuron since no error exists during backpropagation.

The aftermath of this effect in Fig. 2 leads to being each hidden unit in the hidden layer being unable to co-adapt to other hidden units in its corresponding layer. Therefore, each hidden unit would be forced to focus on extracting features which are useful in general.

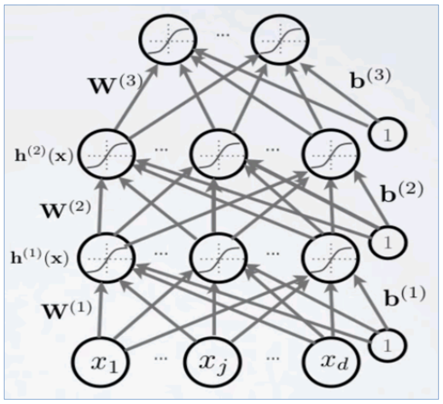


Fig. 1.Multiple perceptron before dropouts

Impact on forwarding propagation equation assuming for each hidden unit, we sample this with binary mask $\mathbf{m}^{(k)}$, where k is an index for hidden layer:

Layer pre-activation for $k > 0$ ($h^{(0)}(\mathbf{x}) = \mathbf{x}$),

$$\mathbf{a}(k)(\mathbf{x}) = \mathbf{b}(k) + \mathbf{W}(k)\mathbf{h}(k-1)(\mathbf{x}) \qquad \text{(eq.1)}$$

Hidden layer activation (k from 1 to L),

$$\mathbf{h}(k)(\mathbf{x}) = \mathbf{g}(\mathbf{a}(k)(\mathbf{x})) * \mathbf{m}(k), \qquad \text{(eq.2)}$$

where $\mathbf{m}^{(k)}$ can be either 1 or 0 with probability 0.5;

Therefore, we are sampling from 2^H different architectures where H is a number of hidden units in network architecture [8]. This shows that a number of architectures that would be sampled are huge. But, since all architecture shares weights during every iteration process, (ie. it has the same weights as it has in other architectures in other iterations) every model architecture is strongly regularized. By sharing weights with other models, the model gets regularized with something that tends to pull weights towards correct values.

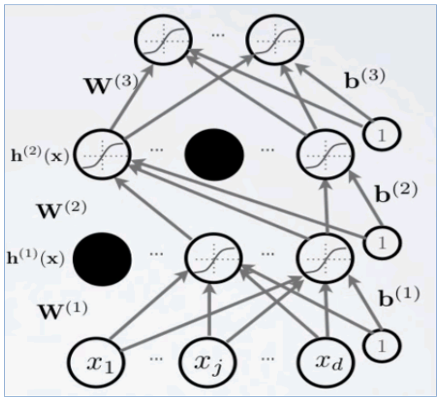


Fig. 2.Multilayer Perceptron after Dropouts

During the testing phase, one option is to sample all architectures and take their geometric mean of output distributions. Another option is to use all hidden units including ones that were dropped out and take half their outgoing weights. The latter method is used in this paper since it retains the fact this exactly computes geometric mean of all predictions of 2^H models would have provided.

B. *Heuristic Genetic Neural Network for Intrusion Detection*

Research has been conducted on joint optimization of three key features [3], ie. in feature selection, structure design, and weight adjustment while these tasks are traditionally optimized separately [9],[10]. These joint optimization works realize the importance of the relationship between critical feature attribute input node and the architecture of an artificial neural network.

In this article, a joint evolutionary neural network structure optimization [3] is tested on XOR dataset and Aegean Wi-Fi Intrusion Dataset. A heuristic based crossover and mutation operator is chosen to help the algorithm not being trapped by local optimum and is able to jump out to search the overall search space. However, the direct nature of encoding is purely random and does not depend on a set of rules, unlike L-system where only a particular number of the synapse can emerge from a specific neuron giving out an overly regularized structure.

One of the main features in the NE algorithm is to protect innovation through speciation. A random or heuristically defined mutation or crossover initially forces the individual architecture of a population to lower its overall fitness level. Therefore, this paper was also extended to protect innovations through mutation or crossover by competing against only its own population topology or niche.

1) *Matrix representation*

A connection matrix as seen in Fig. 3 is used to represent synaptic connections between neurons of all three domains (input neurons, hidden and output neurons).The genotype of each individual in a population is represented by this connection matrix shown in Fig. 3, where its dimension is ***(h+n)*(m+1)***, where ***m*** and ***h*** are the maximum numbers of input nodes and hidden nodes, and ***n*** is the number of output nodes where a 1 or 0 indicates connection of a neuron *j* to a neuron *j*+1.

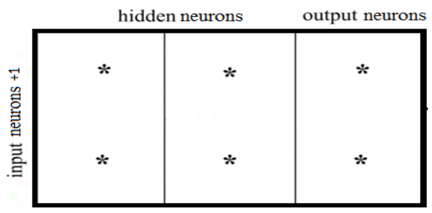


Fig. 3.Connection Matrix

2) *Fitness evalution*

For completion of above procedure, penalty factors are introduced for a number of neuron connections in the topology of an architecture that includes connection of input nodes and hidden nodes. The constraint in a number of connections in network structure gives rise to modularity within the network structure. So, the fitness of each individual from a set of the population is defined as [3]:

$$Fitness(a) =$$

$$\mathbf{80} * (\boldsymbol{accuracy\,rate}(\boldsymbol{a}) + \boldsymbol{\varphi}(\boldsymbol{a}) + \boldsymbol{\xi}(\boldsymbol{a})) \qquad \text{(eq.3)}$$

Where accuracy rate in eq.3 is detection accuracy rate, **φ** is the penalty factor for input nodes, ξ is penalty factor for number of hidden nodes in their topology. Note that accuracy makes only eighty percent of overall fitness.

The detection accuracy rate of the individual ***a*** is defined by:

$$accuracy(a) =$$

$$(True\,positives + true\,negatives(a))/\,sum \qquad \text{(eq.4)}$$

where *true positives, true negatives* in eq.4 are the number of accurate detections, the *sum* is the total number of detections which include both normal instances and abnormal instances.

The penalty factor φ**(a)**, is defined by:

$$\varphi\mathbf{(a)} = \mathbf{10} * (\,\boldsymbol{in}(\boldsymbol{a}) - \boldsymbol{mi}\,) * \boldsymbol{pin}\,/\,\boldsymbol{in}(\boldsymbol{a}) \qquad \text{(eq.5)}$$

where ***in(a)*** is input nodes, ***mi*** is the minimum number of input nodes, *pin* helps to control the influence of a number of input nodes on fitness evaluation in eq.5.

The penalty factor ξ**(a)**, is defined by,

$$= \mathbf{10} * (\boldsymbol{hide}(\boldsymbol{a}) + \boldsymbol{mi} - \boldsymbol{mhide}) * \boldsymbol{pin}/\boldsymbol{hide}(\boldsymbol{a}) \qquad \text{(eq.6)}$$

where ***hide*** is the hidden neurons, ***mhide*** is the minimum number of hidden nodes, ***pin*** helps to control the influence of a number of hidden nodes on fitness evaluation in eq.6. Both penalty factors are assigned only 10 percent of the overall fitness.

3) *Subnet Crossover Operator*

For the process of crossover, a subnet crossover operator is used in of translation of genotype to phenotype. The overall procedure of crossover operator is described as follows:

1. Select randomly a node i ∈ M U H U P.
2. Cross generated subnet of Iteration (j) of node i.

The generated subnet of an input node, ***M*** is the set of its all input connections. Generated subnet of a hidden node is the set of hidden connections, ***H***. Generated subnet of an output node ***P*** is the set of all output connections. Node information constitutes the type of activation function such as sigmoid function to detect a set of possible symmetric data information and make the corresponding network more compact. A compact architecture reduces search in vector space.

4) *Heuristic Mutation Operator*

Adaptive mutation rate is defined by:

$$\boldsymbol{P}(\boldsymbol{g}) = (\boldsymbol{MG} + \mathbf{1}) * \boldsymbol{U} \qquad \text{(eq.7)}$$

where ***g*** in eq.7 denotes current generation of a set of populations, ***MG*** denotes the maximum number of generations for which the population is not optimized continuously, ***U*** denotes a user-defined parameter which is used to control the increase of the mutation rate in the equation. Mutation operator rate increases if ***accuracy***(***a***) is less than or equal to ***accuracy*(*a-1*)** and thus able to rescue from a local minimum and search in a different search space. In [8], mutation operator randomly decides to have weight adaptation, node deletion or node addition. Weight adaptation is done through changing weights using Gaussian noise,

$$\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{N}(\boldsymbol{0}, \boldsymbol{a} * \boldsymbol{P}(\boldsymbol{g})) \qquad \text{(eq.8)}$$

where eq.8 $\boldsymbol{N}(\boldsymbol{0},\boldsymbol{a}*\boldsymbol{P}(\boldsymbol{g}))$ is a Gaussian random variable with mean 0 and standard deviation $\boldsymbol{P(g)}$, $\boldsymbol{w}$ is current weight, $\boldsymbol{P(g)}$ is the adaptive mutation rate defined in, $\boldsymbol{a}$ is a user-defined constant used to scale $\boldsymbol{P(g)}$.

The pseudo code for mutation operator is defined by:

```
1. Calculate Mutation Rate P(g)
  2. While (each individual of population g)
        accuracy(g) <accuracy (g-1)
        If (weight adaptation is selected)
        a connection weight is selected using P(g );
        Else If (node deletion is selected)
        Random node is selected to be removed from
        connection;
        Else (node addition is selected)
        a Random node is selected to be added into
        connection;
```

Individuals are only allowed to compete within their own population of neuron topology of tournament size 3. Only the best individuals in each population are allowed to reproduce and pass its genomes from one generation to another complex generation. The overall evolutionary procedure is described as follows:

```
1. Initialize:
   a) An initial population (tournament size 3) of
      Neuron architecture is generated;
   b) For each generation, a specific topology of
      chromosomes is selected are allowed to compete;
  2. Evaluate: Each individual is evaluated within its
     own class of population topology;

     While (Stopping condition is not met)
        Selection: Select n individual from Parent and
        child individuals to undergo crossover and
        mutation with their own niche;
        Crossover: Perform subnet crossover on
        selected individuals;
        Mutation: Perform mutation over current
        generation individual.
        Evaluate: all individuals are evaluated and
        only the best in the particular topology group
        would be allowed to reproduce.
     End While
```

C. *Neuroevolution Algorithm via Lindenmayer-system with Memory*

Lindenmayer Systems (L-Systems) with memory expands genotypic strings based on production rules creating larger strings based on random DNA bits converted to RNA on valid production rules, ie. a larger phenotype is created from a string of genotypic information. In this paper, another biologically inspired neuroevolution algorithm (NEA) [2] that are structurally modular, hierarchical and recurrent is investigated on XOR and AWID dataset. Lindenmayer-system has recursive nature that gives rise to self-similarity and fractal-like forms. Natural looking organic forms can be generated with L-systems since the recursion grows and becomes more complex. The Lindenmayer System with memory helps to create a structure which is organized, modular and repetitive (multiple uses of the same substructure) and brings hierarchy (recursive composition of the same substructure) in the development of skeleton of the architecture of an Artificial Neural Network. This technique is further compounded to select best DNA and Neural Network architecture by means of an evolutionary process called Genetic Algorithm. The concept of a NE algorithm is classified into direct encoding schemes and indirect encoding schemes. Indirect encoding, the genotype is mapped directly to phenotype and indirect encoding constitutes phenotype mapped indirectly from genotype. Indirect encoding makes recurring ANN structures (modular), minimizes features randomly (regular) and compresses the phenotype leading to smaller architectures that reduce the search space. A fitness penalty is introduced to a number of synapses in each individual of the population that further increases in modularity and generalization. Stanley et al. [11] introduced a hypercube based neuroevolution of augmented Topologies that has a more compact genotype, more scalable and utilizes the geometry of the problem. Grammar Evolution [12] points out to have better minimization of architecture that increases scalability and reduces search space.

1) Neural Structure Codification with Lindenmayer – system

A grammar called L-system with memory is used for the construction of the architecture of an ANN. L-system can be described as a grammar G - { ∑, ∏,**Error! Reference source not found.** } where alphabet is ∑ - { .; f; F; n; [;]; *; B }. The corresponding production rules (∏) are described in Table 1. The starting point of the development process of L-system is called the axiom (**Error! Reference**

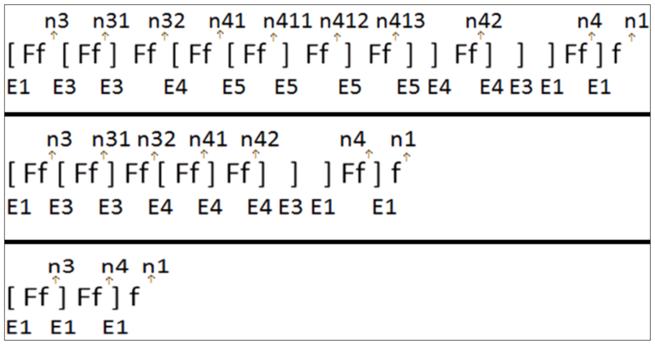
source not found.). The alphabet ‘f’ is a provincial neuron and ‘F’ is the synapse. The symbol ‘[‘ means storing the current state and ‘]’ recovering that particular state. Fig. 4 shows the construction of branches of a tree in step by step iterations. For this particular individual of the generation, the process starts with the axiom (**Error! Reference source not found.** -> .) and second production rule (. ->f) to axiom is applied resulting in f. The genotype iteration process is as follows:

Applying the third rule (3.1) (f -> [f) applied to string f, gives [f. Fourth rule ([-> [Ff]) applied to string [f two times, gives out [Ff]f and then to [Ff]Ff]f. This genotype is shown in Table 2, last row. The string [Ff]Ff] is stored in memory for later reuse.

Table 1. Production Rules (∏) of Parametric L-System with memory

Rule Identifier	Rule
1,2	(1) S -> .(axiom) (2) . -> f
3	(3.1) f -> [f (3.2)f -> [Ff]Ff]Ff]f
4	(4.1) [-> [Ff] (4.2) f ->n (4.3) f -> f B
5	f -> f*

Table 2. Iteration process of String Genotype



```
   n3  n31  n32  n41  n411 n412 n413   n42      n4  n1
[ Ff [ Ff ] Ff [ Ff [ Ff ] Ff ] Ff ] ] Ff ]  ]  ] Ff ] f
E1  E3  E3   E4   E5   E5   E5   E5 E4  E4 E3 E1  E1
```

```
   n3  n31 n32 n41 n42      n4  n1
[ Ff [ Ff ] Ff [ Ff ] Ff ]  ]  ] Ff ] f
E1  E3  E3  E4  E4  E4 E3 E1   E1
```

```
   n3  n4 n1
[ Ff ] Ff ] f
E1  E1  E1
```

In rule (5), (f -> f*) ‘*’ denotes the recovery of saved string from memory. Applying rule (5) in first and second ‘f’ of previous string [Ff]Ff]f further gives out [Ff[Ff]Ff]Ff]]]Ff]f, which indicates that a tree like pair of branches have sprung out from the Neuron 3 and developed into N31 and N32. Applying rule (5) in the second ‘f’ of the previous string means a further extension of the branch from Node N32 into N41 and N42. Applying this sets of rules repetitively and recursively means the same substructure is used multiple times. The recursive composition of that particular substructure gives out the concept of hierarchy.

Applying Rule (3.2) recursively to the string [Ff[Ff]Ff]Ff]]]Ff]f to the fourth ‘f’ of the string gives out [Ff[Ff]Ff[Ff[Ff]Ff]Ff]]Ff]]]Ff]f in Table 2 that represents iteration 8 of phenotype in Fig. 4 where again three branches emerge from N411 neuron.

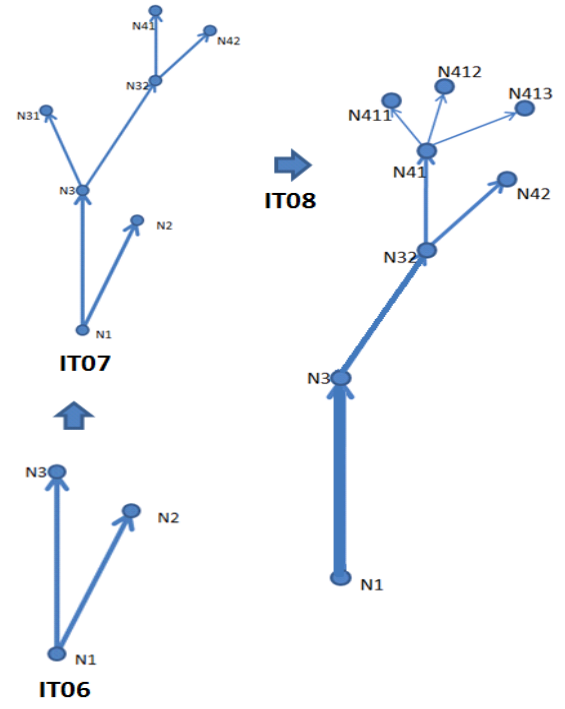


Fig. 4.Iteration Process of Genotype to Phenotype. Only those nodes are kept that have a path from an input to an output neuron.

For the phenotype shown in Fig. 4, output neurons are N411, N412 and N413. It's hidden layer is N41 (first hidden layer), N32 (second hidden layer), N3 (third hidden layer) and the root of the neuron tree acts as the output. It is worth noting, that rule (3.2) and (5) in Table 1 are applied recursively in the string genotype that makes a complex fractal tree like structure demonstrated in Fig. 4. The indirect mapping of genotype to phenotype is shown in Table 2 and its corresponding phenotype is demonstrated in Fig. 4 with step by step iteration. Fig. 4 shows every phenotype’s topological iteration including neuron connections. In the final iteration, only those neurons or nodes are kept which have a link from input nodes to all the way to the output nodes. This indirect mapping of phenotypes reduces search in vector space. Adding a constraint in penalizing fitness of each string with a number of connections and neurons in phenotype ANN topology in each individual in a population provides a more symmetric network structure to modularity.

2) Lindenmayer-Rules extraction with Genetic Algorithm

Each string that is derived through L-Rule extraction, gives rise to mapping to the phenotype of ANN architecture that is subsequently assigned fitness through sufficient iterations. In the quest of making the methodology more closer to biology, recursive rules of Table 2 drives the development of chromosome. The chromosome undergoes crossover and mutation through its iteration lifecycle and the best genotype from several individuals of a population from each generation is passed to the offspring from one generation to another.

Table 3.Function Rule Extraction with Genetic Algorithm controls the translation of Binary String to Production

	00 (U)	01 (C)	10 (G)	11 (A)	
00 (U)	f (UUU)	F (UCU)	n (UAU)	. (UGU)	00 (U)
00 (U)	n (UUC)	. (UCC)	f (UAC)	F (UGC)	01 (C)
00 (U)	F (UUA)	f (UCA)	B (UAA)	f (UGA)	10 (A)
00 (U)	[(UUG)	n (UCG)	[(UAG)	* (UGG)	11 (G)
01 (C)	f (CUU)	] (CCU)	n (CAU)	* (CGU)	00 (U)
01 (C)	* (CUC)	F (CCC)	f (CAC)	F (CGC)	01 (C)
01 (C)	] (CUA)	f (CCA)	* (CAA)	[(CGA)	10 (A)
01 (C)	f (CUG)	* (CCG)	B (CAG)	] (CGG)	11 (G)
10 (A)	* (AUU)	] (ACU)	n (AAU)	f (AGU)	00 (U)
10 (A)	f (AUC))	B (ACC)	f (AAC)	B (AGC)	01 (C)
10 (A)	F (AUA)	[(ACA)	B (AAA)	n (AGA)	10 (A)
10 (A)	* (AUG)	f (ACG)	* (AAG)	] (AGG)	11 (G)
11 (G)	] (GUU)	[(GCU)	F (GAU)	n (GGU)	00 (U)
11 (G)	n (GUC)	B (GCC)	[(GAC)	. (GGC)	01 (C)
11 (G)	f (GUA)	] (CGA)	B (GAA)	F (GGA)	10 (A)
11 (G)	B (GUG)	f (GCG)	* (GAG)	[(GGG)	11 (G)

3) Rules of ANN architecture

Alphabets in Table 2 are a subset of the genetic codes that is generated randomly in bit strings in either ‘0’ or ‘1’. Each 2 bits represent 1 nucleotide, ie. (00,01,10) represents (U,C,A) in the genetic code. 6 bits at a time is read (one chromosome) and is converted to 3 corresponding symbols mentioned in Table 2. A string is allocated for each individual of the population that encodes only a valid production rule reading from the first alphabet to last of the string generated randomly. Shortest valid string that is accepted is .f[Ff*nB], where B is a set of integers and n is its corresponding number. Substrings that do not start with .f[Ff*nB] are rejected for each individual in the population. This process is repeated for all individuals in each generation. The whole process mimics the process of transition from DNA (binary string) to RNA (integer string).

4) Crossover and Mutation Operators

In this article, both the Neuroevolutionary techniques with structural constraints follow the same procedure of crossover and mutation operators. This is done so as to give a robust and clear comparison with each other and with FT based NE algorithm. For each individual in the population, neurons are randomly chosen to be perturbed to another individual of the same population with a crossover rate. For Lindenmayer-system based technique, a specific tree structure is generated for each individual before training starts. The mutation operator sets the mutation rate of each genes for each iteration if the fitness of iteration *j* is less than the previous iteration *j+1*. In this way, the innovations invented from crossover and mutation is protected by allowing it to compete within its own population niche (individuals belonging in same topological form). Mutation operators and crossover operators prevent the algorithm from being trapped into a local minimum and jump out of it by allowing a more cognitive crossover rate and mutation rate. This NE technique adopts tournament selection (*m* individuals) where only *m* individuals would be made to compete with each other instead of overall population. An only best individual chromosome is allowed to reproduce to pass its genomes from one generation to another. Restricting species to compete against its own population niche allows the particular chromosome to survive mutation and crossover effects. The initial fitness of a mutation or crossover might not increase fitness and therefore might force species to become extinct.

An indirect encoding method specifies production rules for genotype to phenotype architecture construction. Production rules can either be neuron layer specifications via connection matrix of hidden, input and output layers or growth rules. So, it is important to make sure the network does not always focus on some specific cluster of neurons and become biased. One of the main drawbacks of this technique is that overly regularized repetitive and regular structures since symbols tend to be rewritten deterministically.

IV. EXPERIMENTS SETUP

D. XOR Problem

XOR is not a linearly separable problem and a machine learning algorithm must be able to differentiate non-linear data in order to solve it. Artificial Neural Network must have hidden neurons connected in a specific manner in order to solve a XOR dataset. So, non-trivial XOR dataset is a good starting point for convergence test of all three techniques before comparing it in AWID dataset.

Both the constraint-based Neuroevolutionary technique is first tested in XOR dataset for comparison and then to AWID dataset. Heuristically based Neuroevolutionary artificial neural network as well as Lindenmayer-system based NE outperforms direct encoded fixed topology based NE (without synaptic constraints) in terms of performance fitness of evolution shown in Fig. 5. Since L-based network structure is highly regular, it has unstable but fast convergence within 14 generations with 3 individuals per generation with thousand iterations each. Heuristic based NE with connection constraints proves to have stable convergence rate with twenty generations whereas fixed topology shown in Fig. 6 implemented in XOR problem is not very symmetric or regular. While dealing with NE evolutionary process, it is found that for the lower number of chromosomes or neurons, the fitness increases rapidly and its corresponding cognitive mutation and crossover rate remains small. For larger chromosomes with a larger number of neurons, fitness often remains static and so mutation and crossover rate increases rapidly.

An iteration example of network visualization via connection matrix representation indicates the individuals' generated are modular and symmetric since both have connection constraints within networks that are shown to evolve rapidly in comparison to fixed topology without constraints that do not generate symmetric or modular network structure. Hundred percent fitness is never achieved since synaptic constrained based NE are always being penalized for using all XOR input features. These visual demonstrations give somewhat an indication of left-right modularity within the networks.

While optimizing input feature selection is not very useful in training XOR dataset, it is very useful when dealing with large datasets which contain many redundant and highly correlated input features that have correlation more than 0.8 with one another. Equation (9) shows an iteration example of Sparse Matrix for Synapse of L-systems based NE and Equation (10) shows an iteration example of Sparse Matrix for Synapse of Heuristic based NE.

$$\boldsymbol{Wi}=\begin{bmatrix}1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1\\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix} \quad \text{(eq.9)}$$

$$\boldsymbol{Wi} = \begin{bmatrix}1 & 0 & 1\\ 1 & 0 & 0\end{bmatrix} \quad \text{(eq.10)}$$

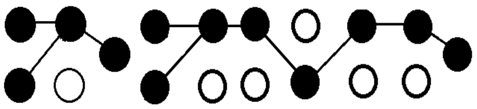


Fig. 5.Left: Structure of an individual Heuristic based NE of eq.9. Right: Structure of an individual L-systems based NE of eq.10. Both correspond to connection matrix equations above.

From Fig. 5, it can be deduced that FT based direct encoding scheme without synaptic penalty NE technique's evolution process is unstable and gives a slower convergence rate ie. 45 generations with tournament selection size 3. It's fixed topology for each generation prevents it from node addition and deletion. This drastically increases the vector search space in contrast to modular structures. Modular networks provide a modular and compact structure that increases scalability. The dimension of connection matrix equation shown for L-systems based NE is [{(8*2)+1} * 3], where 8 is a number of hidden layers and 2 is input feature. The sparse connection matrix for heuristically based NE is [{(1*2)+1} * 3], where 1 is a number of hidden layers and 2 being the input features shown in Fig. 5.

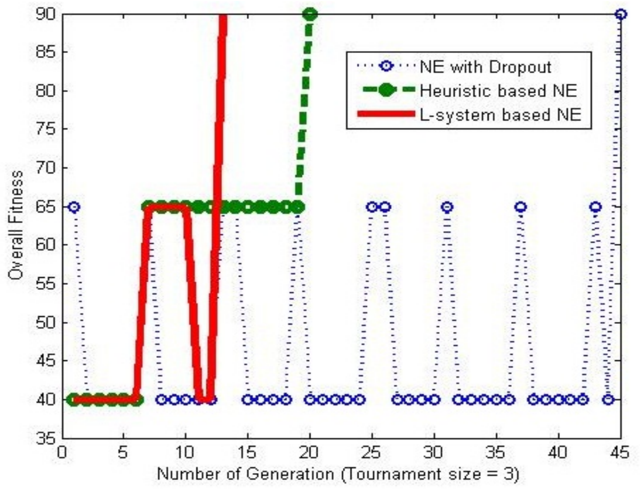


Fig. 6.Fitness of NE-based Direct Encoding scheme with Dropout Compared with Indirect Encoding scheme (L-system based NE)and NE-based Direct Encoding scheme (Heuristic NE)

E. AWID dataset

According to the type of used pattern, intrusion detection techniques can be classified into two categories: misuse detection and anomaly detection [1][6], Misuse detection, which is a rule-based approach, uses stored signatures of known attacks to detect intrusions. This approach can detect known intrusions reliably with the low false positive rate. However, it fails to detect novel attacks, and the signature database has to be updated manually for future use when new attacks occur. Anomaly detection uses normal patterns to model intrusions, and any deviation from the constructed models for normal behaviors is considered as anomalies. However, it is very difficult to precisely model all normal behaviors, and it is easy to mistakenly classify normal behaviors as attacks, which will result in high false positive rate. Therefore, the key to anomaly detection is to select an appropriate model for normal behaviors.

A publicly available dataset compiled by [1] is very beneficial for research in Wireless Intrusion Detection Systems called as Aegean Wi-Fi Intrusion Dataset (AWID). The dataset consists of four classes that include flooding, injection, and impersonation, and the normal class. It consists of 156 attributes representing WLAN frame fields as well as physical layer data. Of all the attributes, noise data is also present among other useful data.

All features in the dataset consist of either MAC layer information such as Source Address (wlan_sa), Destination Address (wlan_da) or Radiotap information such as Signal Strength and Packet Number (frame_number).SSID feature has string values and MAC addresses having hexadecimal values are converted to integer values while processing the dataset. For example, a MAC address has a value of 54868889197 but signal strength consists of -33 dB. Therefore, it is always better to normalize the dataset ranging from 0 to 1 before implementing on any the neuroevolution technique.

Training samples of reduced dataset used in this paper have a total of two million rows and a manual reduction of features from 156 to 11 features. This is done in order to deal with missing data rows that most of the samples consists with and attributes that had unique values or mostly zeros. Additional attributes with high correlation coefficient (greater than or equal to 0.8) are also cut off from themain dataset. Highly correlated attributes influence each other and bring little information. As a result, it is not beneficial to maintain them in the dataset. Values were normalized to the values assumed in the interval [0,1]. Normalization is necessary in order to provide the data features with the same order of magnitude. These changes were made in the dataset based onthe proposed changes by [6].

Table 4.Dataset reduced to 11 features

1. Destination Address.
2. Sub-type is identify
3. Transmitter Address
4. Duration
5. wep.iv
6. Type
7.Bid
8. Source Address
9. RSS
10.Protected
11.wep.key

Few major drawbacks of this dataset are that the attacks launched in this data are by specific penetration testing tools that build patterns of intrusions which are easily detectable. On a real scenario, an attacker might use non-existing or novel tools to bypass the intrusion detection system. Moreover, the dataset does not consider the mobility if the attacker.

V. Results And Analysis

Iteration process of a Neuroevolution technique from simple to dense and then to a more compact network structure is done together with weight mutation. Following the work of [3], more emphasis is given to detection rate compared with connection cost. This is done by assigning a lower percentage (10%) of fitness to connection cost and input connection cost (10%).

As mentioned previously, a dataset can be called regular if it has repeated samples and is set to have modular decomposability if it can be divided into a set of smaller subsets each independent of one another. This division can be recursively used to make further decomposition of subsets. Since the above mentioned direct and indirect encodings NE techniques with connection constraints are ideal to solve modular and regular datasets, this investigation of using AWID dataset into NE algorithm with flexible number of neurons with connection costs and fixed topology based non-flexible number of neurons without connection cost gives a robust view of behavioral comparison between symmetrical or modular structures and its counterparts.

Table 5.Comparison with all three NE algorithms

Method	Num of Input features	Number of hidden nodes	Detection rate(%)	False positive rate(%)
Dropout Based NE	26	32	76.47	6.83
Indirect Encoded L-sys NE	6	10	82.12	4.54
Direct Encoded Heuristic NE	9	8	86.62	3.5

As mentioned previously, both heuristically based NE and L-system based NE optimize feature selection as well as network structure and weights simultaneously. But, the direct encoding technique with fixed topology only optimizes weights randomly without any mutation or crossover rates. These features of constraint-based NE show good generalization detection rates and false positives during testing data. Using L-systems to generate ANN with production rules ensures searching more network structure by placing chromosomes in alternate positions within chromosomes to search the best structure.

In comparing test results, heuristic based NE gives best detection accuracy and lower false positive than the overly regularized L-system based NE giving an indication that dataset used is not very regular while direct encoding based

NE gave lowest detection rate and higher false positive rate in comparison with indirect encoding technique L-system based algorithm. This is because it takes into account all input features together with redundant ones while developing the neural network architecture and testing phase which proves the fact that some features in attributes reduced dataset are also redundant. These results further prove that optimization of input feature selection, weight optimization, and neural network architecture have joint contributions in Artificial Neural Network in achieving faster detection rate and false positives. From Table 5, it is shown that best result with 91% Detection Rate and False Positives of 7.7% was achieved by specification 10,48,1 with 10 being input features and 48 being a number of hidden neurons on L-system based Neuro-Evolution technique. Fitness formula used in this article always ensures the smallest number of hidden neurons within an ANN structure is used giving a compact structure of the overall network and improves scalability.

Using the Heuristic based Neuro-Evolution algorithm with Connection constraints of in AWID dataset gives the best results while the Indirect Encoded highly regularized L-based NE algorithm does not perform as well as the previous one. This indicates that Aegean WiFi Intrusion Detection dataset can be regarded as regular and symmetric to some extent. The worst result is given by fixed topology based direct encoding scheme without connection constraint which further indicates that the dataset is regular and symmetric.

VI. Conclusion

A major drawback of regular and symmetric network infrastructure is that it tends to get biased towards regularity of data. Henceforth it would not be able to classify data if the dataset during testing phase suddenly shifts to being regular.

Future works to be considered is comparing Neuroevolution algorithms to a more regular and modular dataset such as Retina problem, Hierarchical XOR problem and then on AWID data using well-known modularity Q-score [13] formula to measure modularity of networks after implementing the NE techniques on Retina Problem as well as AWID dataset. To calculate regularity, best individuals' network infrastructure of nodes of each NE technique would be compressed using another popular formula called Lempel-Ziv-Welch compression algorithm [5]. A visual demonstration of network placementof nodes via python software while implementing it on AWID dataset would also give a better judgment of its network modularity and regularity. In this way, it is possible to mimic the structures of human brains to make more intelligent neural networks. Developing a better structured, modular and compact artificial neural network is a step towards this goal.

VII. Acknowledgemnt

I am very grateful to Prof. Dr. Ken Ferens for his graduate coursework on Applied Computational Intelligence that has provided me a very clear understanding of neural networks and the challenges faced by traditional algorithms. Further, I am very thankful to Muhammad Salman Khan, who is a Ph.D. candidate under the supervision of Prof. Dr. Ken Ferens who has helped me with ideas and also steered the current idea of this paper

References

[1] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 184–208, 2016.

[2] L. M. L. De Campos, R. C. L. De Oliveira, and M. Roisenberg, "Evolving Artificial Neural Networks through L-system and evolutionary computation," *Proc. Int. Jt. Conf. Neural Networks*, vol. 2015–Septe, 2015.

[3] B. Zhang, "A Heuristic Genetic Neural Network for Intrusion Detection," *2011 Int. Conf. Internet Comput. Inf. Serv.*, pp. 510–513, 2011.

[4] H. Lipson, "Principles of modularity, regularity, and hierarchy for scalable systems," *J. Biol. Phys. Chem.*, vol. 7, no. 4, pp. 125–128, 2007.

[5] J. Mouret, U. Pierre, M. Curie-paris, and J. Clune, "Evolving Neural Networks That Are Both Modular and Regular : HyperNeat Plus the Connection Cost Technique Categories and Subject Descriptors," 2014.

[6] B. Alotaibi and K. Elleithy, "A Majority Voting Technique for Wireless Intrusion Detection Systems," 2016.

[7] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs," vol. 2005, 2005.

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," pp. 1–18.

[9] B. Zhang and X. Jin, "A Joint Evolutionary Neural Network for Intrusion Detection," pp. 5–8, 2009.

[10] E. Michailidis, S. K. Katsikas, and E. Georgopoulos, "Intrusion Detection Using Evolutionary Neural Networks," *2008 Panhellenic Conf. Informatics*, pp. 8–12, 2008.

[11] J. Gauci and K. O. Stanley, "Autonomous Evolution of Topographic Regularities in Artificial Neural Networks," vol. 22, no. 7, pp. 1–38, 2010.

[12] I. G. Tsoulos, D. Gavrilis, and E. Glavas, "Neural

Network Construction using Grammatical Evolution," no. 1, pp. 0–5.

[13] M. E. J. Newman, "Modularity and community structure in networks."