

Chapter 2

Neuromorphic Computing for Cognitive Augmentation in Cyber Defense

Robinson E. Pino and Alexander Kott

2.1 Introduction

The growth of digital content and information through the World Wide Web is increasing rapidly and more of this traffic is generated by smart mobile low size, weight, and power (SWaP) devices that are constantly sending/receiving information to/from the network for up-to-date operation. In terms of data, according to an IDC report by Gantz and Reinsel in 2012 [1], from 2005 to 2020, the digital universe will grow by a factor of 300, from 130 to 40,000 exabytes, and from now until 2020, the digital universe will about double every 2 years. The size of the digital universe in 2010 was estimated at 1,227 exabytes [1] in particular. Therefore, it can be expected that an increasing number of low SWaP devices will be implemented to offer enhanced functionality in terms of the complexity and number of services offered to users within the physically and electronically constrained form factor architecture. From a network security stand point, it will be important for the Army to ensure security and trust in the operation and functionality of smart mobile tactical devices. However, from the user's point of view, performance degradation due to security add-ons may degrade device performance during operation and during operations where speed is critical, enhanced security could degrade operational effectiveness. Therefore, it is the main goal of this effort to perform basic research in methods and techniques to provide security to mobile tactical networks while ensuring low SWaP technical requirements for operation. In this pursuit, we have considered two basic research areas that could provide a revolutionary solution to

R.E. Pino (✉)

U.S. Department of Energy, Office of Science, Washington, DC 20585, USA
e-mail: robinson.pino@science.doe.gov

A. Kott

Network Science Division, U.S. Army Research Laboratory, Adelphi, MD 20783, USA
e-mail: alexander.kott1.civ@mail.mil

the problem. The first technology area is memristor-based computing and the second area is artificial neural networks. It is expected that memristor-based physical computing architectures will deliver ultra-low SWaP and neural networks will enable parallelism and reconfiguration benefits. This chapter will provide a brief overview of the memristor technology and its applications within neural networks and their potential application to enabling human cognition augmentation in the Cyber-domain.

2.2 Memristor Based Technology

Interest in memristor-based technologies resurfaced in 2008 with the publication of two papers by Strukov in May 2008 [3] titled “The missing memristor found” and followed by Williams in December 2008 [4] with a paper titled “How We Found the Missing Memristor”. These papers reference the pioneering early research work by Chua in 1971 [2] in which the memristor device is theorized to exist based on physical principles. The memristor is a two terminal passive device whose name is the contraction for memory resistor whose intrinsic property is to remember its previous state. We can think of the memristor as a variable resistor device whose resistance or impedance value can be changed by electrical current pulses; the electronic equivalent to the mechanical potentiometer. For example, positive current can decrease the resistance value while negative current can increase the resistance value. In addition, the memristor device exhibits electronic hysteresis which means the device current-voltage properties follow a characteristic loop behavior as shown in Fig. 2.1 [5]. From the figure, we can observe that the device behaves linearly (between -0.35 and 0.22 V approximately) and non-linearly (below -0.35 and above 0.22 V approximately). In the linear region, we can observe two states of high and low resistance. The non-linear region is the programming region that allows the device to move from a high resistance state to a low resistance state and vice-versa within the hysteresis loop as shown in Fig. 2.1 and described in more detail the published literature [5–7].

The promise memristor-based technologies offer to network security applications could be revolutionary. According to Williams [4], the potential for memristor-based applications goes towards designing new circuits that mimic aspects of the brain where neurons are implemented with transistors and synapses with memristors [4, 8]. Recently, Shevenell [10] characterized the collection performance of FPGA-based sensors compared with the current commodity server-sized sensors to be over 30 times higher collection rates with 150 times less weight and less than 4 % of the size as shown in Table 2.1. Also, Xia and his team [9] published a paper showing that memristors could improve field-programmable gate arrays (FPGA) by shrinking the processor by nearly a factor of 10 in area. Given Shevenell’s and Xia’s research, a memristor-based FPGA sensor could outperform the current state-of-the-art systems while reducing the form factor by 10 times. This new type of physical-memristor-based neural network computing technology could be programmed or trained to

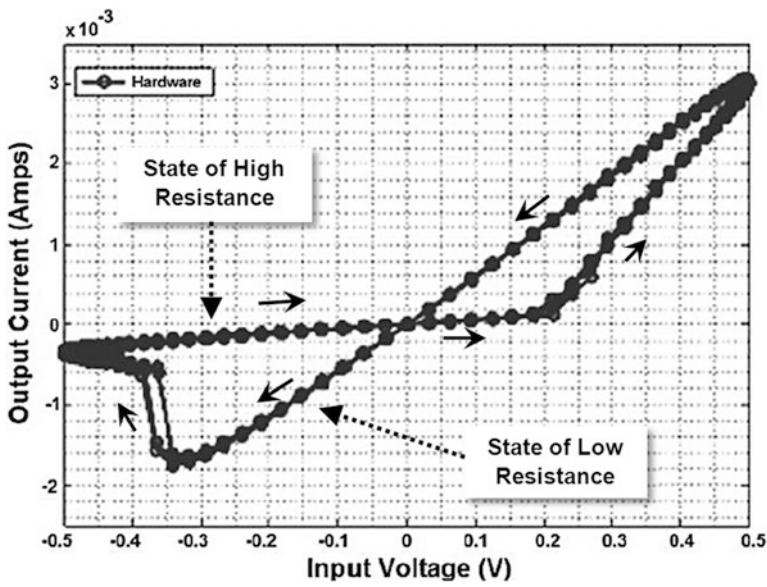


Fig. 2.1 Example of memristor device electronic hysteresis characteristic behavior, adapted from [5]

Table 2.1 Comparison of size, weight, power and throughput [10]

	Dell Power Edge R715	FPGA Xilinx ZYNQ	Raspberry PI
Weight (lbs)	49.6 ^a	0.321	0.126
Size (in)	29 × 17 × 3.4 ^a	8 × 8 × 1	4 × 3 × 1
Power (W)	388 ^a	3–6	3.5–7
Collection Rate (Mbps)	300 ^b	10,000+	66

Information obtained from obtained from ^a[12] and ^b[11]

perform the data analysis within a network intrusion detection system (IDS) at the sensor with enhanced performance and SWaP capabilities. Table 2.1 also compares the SWaP characteristics of the Raspberry Pi [13], a credit-card sized computer that can be used similar to a desktop PC, for spreadsheets, word-processing, games, and high-definition video.

Physical implementations of memristor-based neural networks have been already explored and characterized by Pino and his team [14–16]. For example, Pino describes an apparatus and method for dynamically and autonomously causing a circuit to reconfigure itself and produce a different output for the same input relative to the circuit’s initial state; and the circuit’s state remains constant until the memristor’s resistance is changed, at which point the circuit’s function is “reprogrammed” [14]. In addition, it has been shown how a physical memristor-based computing architecture can perform the function of neurons and synaptic connections by providing variable resistance circuits to represent interconnection strengths between

neurons and a positive and negative output circuit to represent excitatory and inhibitory responses [15]. Also, Pino and Bohl [16] have demonstrated that analog resonant logic self-reconfiguration can be achieved without physical re-wiring where components only include passive circuit elements such as resistors, capacitors, inductors, and memristor devices. And recently, by leveraging only the binary memristor states of ON(low resistance)/OFF(high resistance), Pino and Pino [17] demonstrated that reconfigurable computing logic can be implemented by a decoder to select memristor devices whose ON/OFF impedance state will determine the reconfigurable logic output, and the resulting circuit can be electronically configured and re-configured to implement any multi-input/output Boolean logic computing functionality. In terms of power consumption, memristor-based architectures would yield significant improvements on the order of 20×10^{-15} J [18] compared to CMOS devices (about 60×10^{-15} J [19]) per operation at the 45 nm process technology node. The one key difference between CMOS-only and memristor-hybrid architectures is that memristor-based technologies would theoretically require no stand-by-power which is one of the main sources of energy consumption in conventional computing architectures. Another important difference is the complexity of connectivity in which the memristor device has 2-terminals [2] compared to the 4-terminals (source, gate, drain and body) in the field effect transistor that need to be properly biased for operation. While these are promising developments showing that it is possible to manufacture the technology, for very large-scale implementations of designs containing over 1,015 networked memristor devices, it will be a benefit to reduce the programming overhead of each memristor device. With this end in mind, we have developed in-house a basic neural network algorithm model called ALIEN (for Adaptive Locally Influenced Estimation Network) that simplifies simulation and implementation of memristive hardware designs while being able to demonstrate the direct applications to network security presented and discussed in the results and discussion section of the report. Appendix B shows the memristor characterization results achieved by the Army Research Laboratory in collaboration with the Air Force Research Laboratory.

2.3 Artificial Neural Networks

The memristor device's behavior of exhibiting discrete states of varying resistance is intuitively reminiscent of synaptic weights in a neural network and offers vast potential to the eventual implementation of hardware-based neuromorphic computing architectures with very modest form factor, size, weight, and power dissipation characteristics. And while neural networks are not a new construct in the field of computer science, we have developed a simple neural network learning model named ALIEN that seeks distinction by being designed from the ground to facilitate its implementation in memristive hardware. In order to accommodate the nondeterministic factors (manufacturing variability, inexact model response, etc.) currently associated with memristor production processes [20], the ALIEN learning algorithm

eschews more sophisticated update schemes such as Backpropagation and instead achieves gradient descent with a rudimentary objective function (equivalent to 1 or -1 depending on the sign of the difference between a desired output and the actual output, or 0 if the actual output was correct) with a completely randomized learning rate as will be discussed in detail in the algorithm description section.

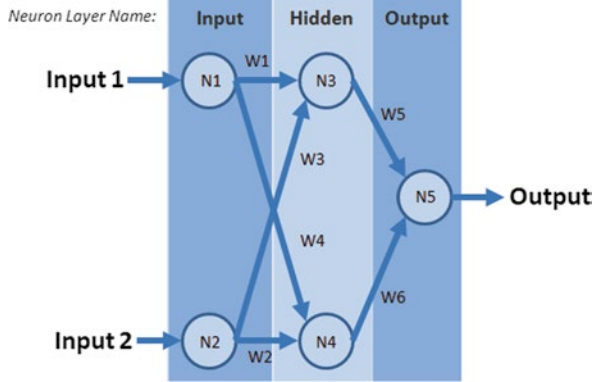
Artificial neural networks are attractive for their ability to adapt and learn to distinguish or classify patterns. In particular, the solution to a pattern separation problem is basically a problem of obtaining a criterion for distinguishing between the elements of two disjoint sets of patterns [22]. For example, if the patterns can be represented by points in a Euclidean space, one way to achieve separation is to construct a plane or a nonlinear surface such that one set of patterns lies on one side of the plane or the surface, and the other set of patterns on the other side [22]. Towards this end, multi-layered feed forward neural networks can achieve nonlinear separation by applying successive linear computing operations [23, 24]. Mangasarian [22] and McCulloch and Pitts [21] have shown that both linear and nonlinear separation can be achieved by linear programming. In generic terms, neural networks offer the potential to generalize a problem by learning to detect patterns from data using a parallel distributed computing architecture, and the key to program, train, or teach a neural network to perform such a function is the learning algorithm. In the published scientific literature, there are several learning algorithms that can be used to train neural networks; however, from a pragmatic implementation perspective, the Backpropagation algorithm has been used for several decades as the de-facto learning algorithm to train and prototype multi-layered neural network-based systems [23, 24]. Recent neural network research and development employs advances in computational methods and mathematical techniques to achieve fast convergence rates and accuracy; however, it is the goal of our in-house research to simplify this approach to map a low mathematical complexity neural network learning algorithm for applications with physical memristor-based neural networks that will minimize circuit overhead in design and manufacturing while preserving the adaptive properties of artificial neural networks.

2.4 ALIEN Learning Algorithm

The main purpose of the ALIEN learning algorithm is to minimize the number and complexity of computations required to train a neural network. Figure 2.2 shows a two input and one output neural network with a single hidden layer, and this will be used to explain the ALIEN learning algorithm.

The feed forward operation of each neuron, N_n , is that of a threshold summing node in which whenever the summed product of the input times the weight, W_n , over all inputs m is greater than a given threshold value, V_{th} , the output of the neuron will be 1 otherwise -1 as shown in Eq. 2.1. Therefore, applying this feed forward operation from layer to layer will result in an output. From this formalism,

Fig. 2.2 Feed forward neural network configuration with two inputs, two hidden and one output (five neurons) and six weights or synaptic connections



we can see that the only controlling parameter to change the neuron's output is the weight element, W_n .

$$\text{Neuron Output, } N_n = \begin{cases} 1 & \text{if } \sum_{n=1}^m W_n \times \text{Input}_n \geq V_{th} \\ -1 & \text{otherwise} \end{cases} \quad (2.1)$$

The algorithmic operation to update each weight element, W_n , is as follows:

Compute output neuron direction (OND) by taking the sign of the subtraction between the desired output and actual output of the network. The output neuron direction is the polarity of the output of the neuron which needs to be corrected up or down (plus or minus).

$$\text{OND} = \text{Desired Output} - \text{Actual Output} \quad (2.2)$$

Compute hidden layer neuron direction (HLND) by taking the sign of the multiplication between the output neuron direction and connecting weight element. For more than one output to which the hidden neuron is connected, compute the sign of the total sum for all output neuron directions times the weight connecting the hidden neuron to any output neuron.

$$\text{HLND} = \sum_{n=1}^m \text{OND}_n \times \text{Weight}_n \quad (2.3)$$

Compute change value to output weights (ΔOW), which connect the hidden layer to the output layer neurons, by multiplying: output neuron direction, learning rate, random value and hidden layer neuron output (HLNO). The learning rate is any positive number normally ranging from 10 to 0.00001, zero means not learning and smaller value means slower convergence. The random value is any value between 0 and 1.

$$\Delta OW_n = \text{OND}_n \times \text{LR} \times \text{RAND} \times \text{HLNO}_n \quad (2.4)$$

Compute change value to hidden weights (ΔHW), which connect the inputs to the hidden layer neurons, by multiplying: hidden neuron direction, learning rate, random value and input to the hidden neuron or input layer neuron output (ILNO).

$$\Delta HW_n = HND_n \times LR \times RAND \times ILNO_n \quad (2.5)$$

Update weights by adding the weight change value to each synaptic weight element.

$$W_n = W_n + \Delta W_n \quad (2.6)$$

During our discussion on learning and training of the neural network two important terms will be mentioned iteration and epoch. Training or learning iteration is the process of applying the ALIEN learning algorithm to update the synaptic weights in the neural network for the entire training dataset once. The epoch is the process of generating new random synaptic weights for all synaptic weights in the neural network.

In order to validate ALIEN's feasibility as a machine learning algorithm, we sought to demonstrate its ability to correctly learn the two-bit Boolean exclusive OR (XOR) function and 8-bit odd Boolean parity. These results are shown in Appendix A. Then, we implemented ALIEN to solve two network security challenges: First to classify malicious traffic from network flows; and second to classify A or MX DNS record requests based only on the network packet data. For context, let's take a look at how neural networks have been applied in network security applications.

2.4.1 Learning Algorithm Characterization

2.4.1.1 XOR Function

The initial experiment to demonstrate the capability of ALIEN was to learn the function of the XOR Boolean function as shown in Fig. 2.3. The importance of the XOR function is that it is a simple two-input and one-output non-linear classification problem. Therefore, it is important to demonstrate the neural network can perform this basic non-linear classification problem by learning to perform the XOR function perfectly.

The results of the trained ALIEN neural network are shown in Fig. 2.4. The figure displays the percentage of successful convergence and not successful convergence versus learning rate. The maximum number of iterations allowed during the experiment was 10,000 iterations. Whenever the training process took in excess of 10,000 iterations, the experiment was determined to be not successful. From the Figure, we can observe that for low learning rates, below 0.001, the percentage of successful convergence is below 50 % and becomes lower with lower rates. Figure 2.5, shows the average and median number of iterations required for the network to converge to a perfect solution. The total number of epochs was 50 with

Fig. 2.3 The complete XOR function input and output data set

Input A	Input B	Output C
0	0	0
0	1	1
1	0	1
1	1	0

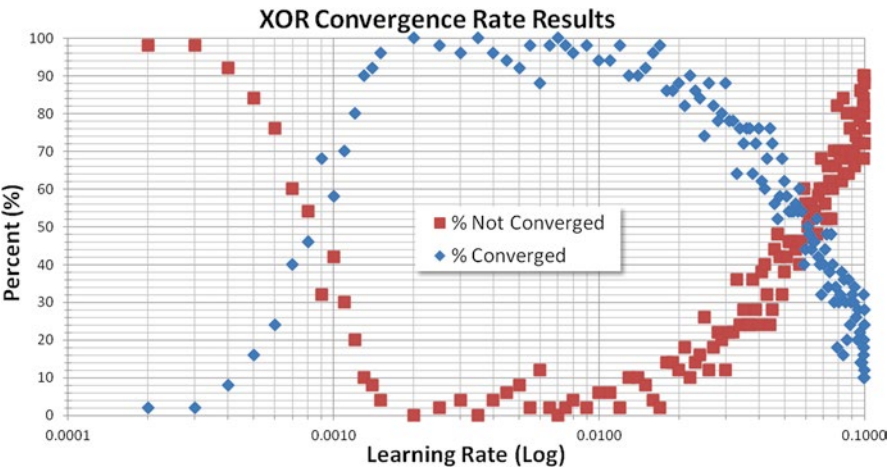


Fig. 2.4 XOR function percentage convergence versus learning rate (in log x-axis scale)

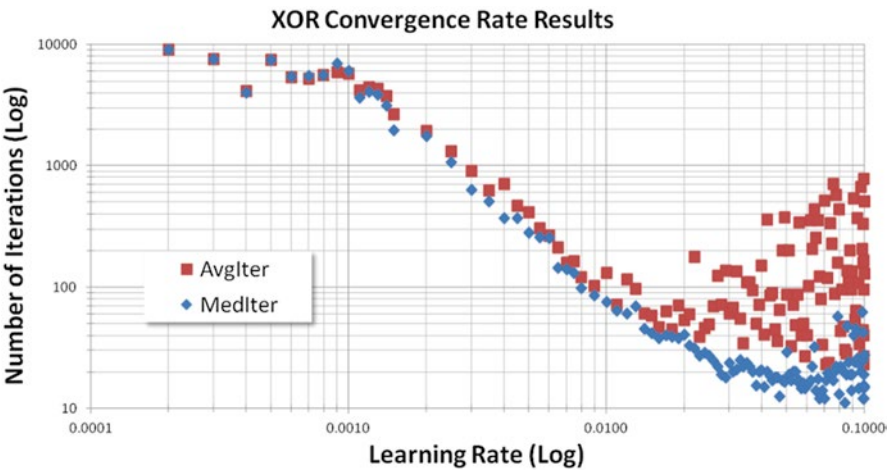


Fig. 2.5 XOR function number of average (*square*) and median (*diamond*) iterations required for a successful convergence versus learning rate (in log x-axis scale)

four hidden neurons and one output neuron. From the Figure, we can observe that as the learning rate was increased, the median number of iterations required to achieve a perfect solution were lower while the average number of iterations became noisier. From these results, we can conclude that for small learning rates, the number of iterations required to achieve the perfect solutions increases as the synapses converge slowly toward the solution. Conversely, as the learning rate becomes large, the median number of steps to achieve a perfect solution is reduced; however, the lower number of iterations required comes at the cost of increased likelihood of failing to reach a comprehensive solution within a training epoch. With a large maximum learning rate, the network may continually find itself on different sides of the ideal configuration because its adjustments lack the granularity to settle down in the global error minimum.

Figure 2.6 shows the percent successful and unsuccessful as a function of learning rate in a linear scale. From the figure, it is easy to point to the region yielding a 50 % success rate of converging to a perfect solution between learning rates from 0.05 to 0.07 approximately. In addition, we can observe that as the learning rate becomes larger, the distribution of successful and not successful convergence grows and widens as the learning rate grows. Figure 2.7 shows the average and median number of iterations required to achieve successful convergence. In particular, it is interesting to note that from an approximate learning rate between 0.02 and 0.1, the median iteration rate stays consistently below 100; while on the other hand, the average number of iterations required grows to an upper bound of close to 1,000 iterations. In general, from this experiment, we can determine the most suitable parameters to achieve reliable convergence when performing the two-input non-linear classification problem of the XOR Boolean function. From Fig. 2.6, we can

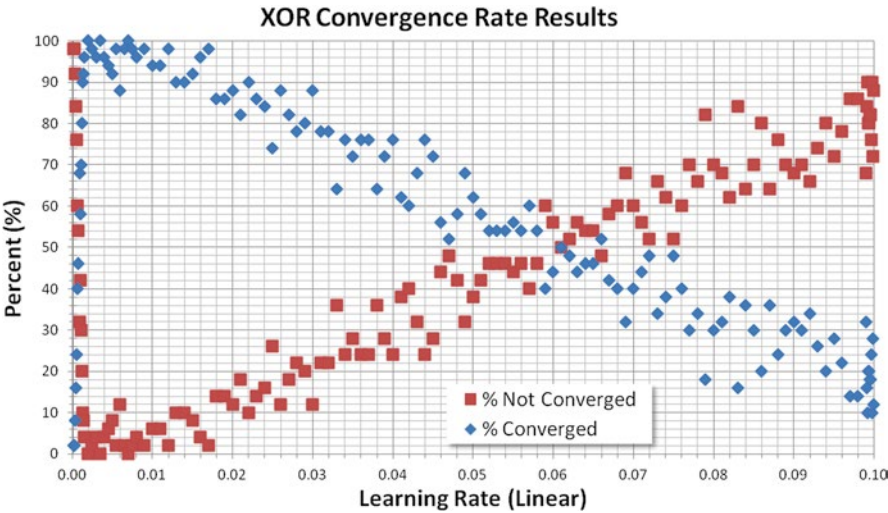


Fig. 2.6 XOR function percentage convergence versus learning rate (in linear x-axis scale)

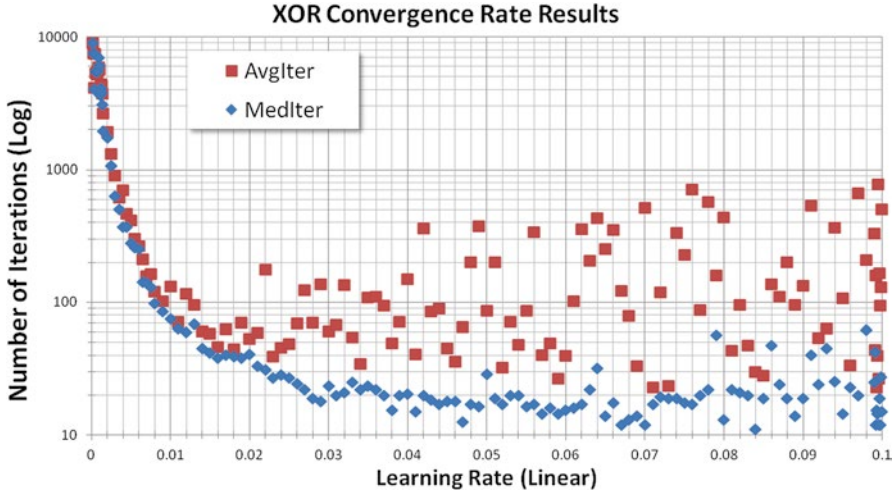


Fig. 2.7 XOR function number of average (*square*) and median (*diamond*) iterations required for a successful convergence versus learning rate (in linear x-axis scale)

see that at a learning rate of 0.02, we could achieve a 90 % convergence rate that would require approximately 100 iterations to achieve perfect learning.

From our experimental results, we found that the ALIEN model is very capable of perfectly learning the 2-bit Boolean XOR function shown in Fig. 2.3 as well as the complement of the function; that is, the inverse of the output. One interesting aspect however was to characterize and explore the convergence space of ALIEN given its simple learning architecture and design. From our experiments, we discovered that the number of required training iterations before arriving at a solution was sensitive to the value of its initial weights (the current application randomizes all synaptic weight values upon initializing the neural network). The fully-connected neural network architecture we employed for this experiment included 2 inputs, one hidden layer of 4 neurons, and 2 outputs (one to emit the XOR result and one to emit the complement of the XOR function). In some trials, ALIEN was able to perfectly learn the two functions in less than 10 training iterations. However, in other trials, 1,000 iterations were not enough to generate the desired output vector.

2.4.1.2 8-Bit Odd-Parity Classification

Another basic experiment to demonstrate non-linear classification was to demonstrate ALIEN's ability to learn and perform 8-bit Odd Parity classification. The idea of parity classification is simple as shown in Fig. 2.8. From the figure, we can see that the goal is to count the number ones and determine if the total number of one's are odd or even. In our experiment, we trained a neural network with ALIEN to determine if the input vector presented was odd or not. Therefore, we constructed an

Bit-1	Bit-2	Bit-3	Bit-4	Bit-5	Bit-6	Bit-7	Bit-8	Parity
0	0	0	0	0	0	0	0	Even
0	0	0	0	0	0	0	1	Odd
0	1	0	1	0	1	0	1	Even
0	0	1	0	1	0	0	1	Odd

Fig. 2.8 Examples of 8-bit even and odd parity

ALIEN neural network containing 8 inputs, one hidden layer containing 256 neurons, and one output. The result was that ALIEN was capable of reliably determining 8-bit odd parity with perfect accuracy.

The network was able to perfectly emit correct odd bit parity within just 478 training iterations in some trials while in others up to 1,675 iterations were required. As with the 2-bit XOR implementation, this variance suggests that required iterations are highly sensitive to the random synaptic weights at the beginning of the training epoch as it is expected given the random design nature of the initialization process. Therefore, we performed another set of characterization experiments to determine the behavior and suitability of the ALIEN neural network learning algorithm to perfectly learn the 8-bit odd parity function. Similarly to our previous experiment, the maximum number of iterations was set to 10,000, and the total number of epochs performed was seven given the limitations of our simulation computing platform.

Figure 2.9 shows the percent rate of successful and not successful convergence to learning the function perfectly as a function of learning rate (x-axis in log scale). From the figure, we can observe that for low learning rates the successful convergence rate is at or very close to 100 % consistently. Figure 2.10 displays the average and median number of iterations required to achieve perfect convergence. From the figure, it is shown that for learning rates between 0.00001 and 0.0001, the average and median number of iteration required to achieve perfect convergence is lower than 3,000. In particular, we can observe that the number of iterations reduces as the learning rate increases.

Figure 2.11 shows the percentage of successful and not successful convergence versus learning rate in linear scale (x-axis). From the figure, we can observe that the region that yielded a 50 % convergence rate is between 0.0006 and 0.0008. In particular, the figure also shows how quickly the successful convergence rate drops exponentially towards zero as the learning rate reaches 0.001. Figure 2.12 displays the average and median number of iterations required to achieve perfect convergence versus learning rate in linear scale (x-axis). From the figure, it is clear that the number of iteration jumps to very close to 10,000 required iterations as the learning rate increases from 0.0002 to 0.0008. In general, the optimum parameters for ALIEN to train a neural network to perform 8-bit odd parity can be determined to be at learning rate of about 0.0001 that would require an estimated 1,000 iterations .

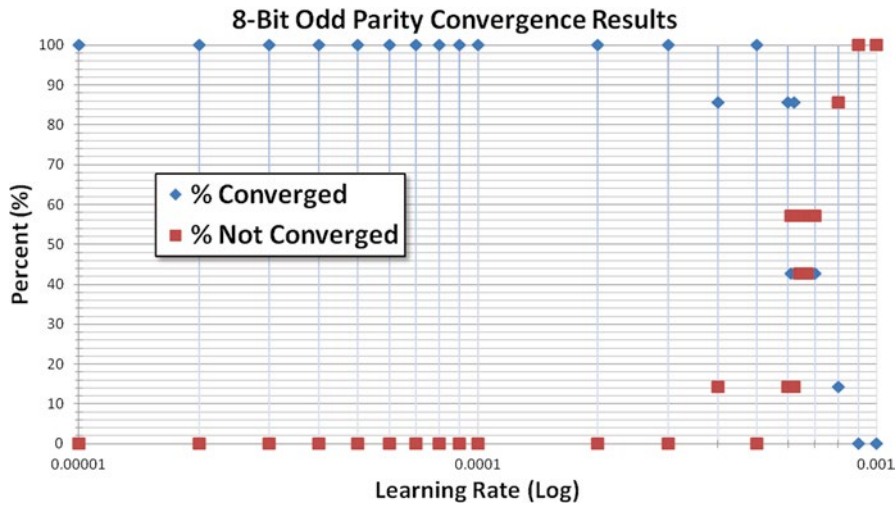


Fig. 2.9 8-Bit ODD parity function percentage convergence versus learning rate (in log x-axis scale)

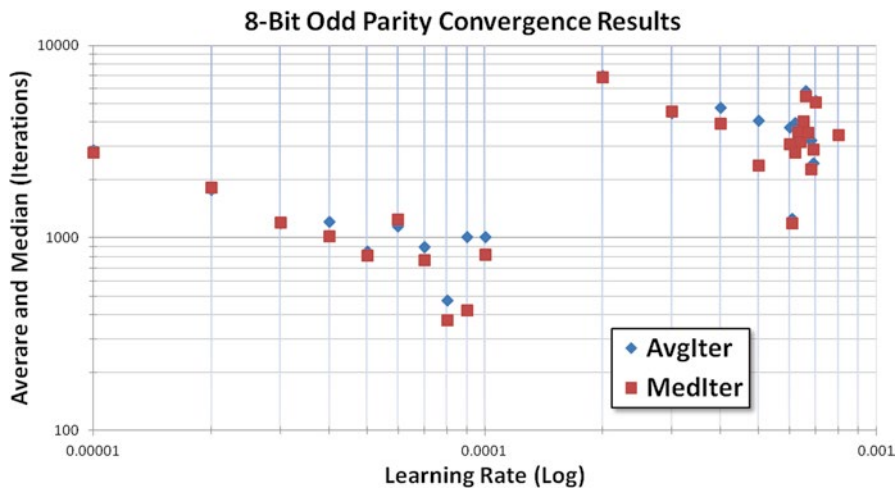


Fig. 2.10 8-Bit ODD parity function number of average (*diamond*) and median (*square*) iterations required for a successful convergence versus learning rate (in log x-axis scale)

Finally, we characterized the performance of the neural network to classify in the event that it was trained with missing or partial information. Therefore, we designed an experiment with the 8-bit odd parity data set to characterize the response. The experiment performed consisted of 70 epochs, a maximum of 2,000 iterations, at a fixed learning rate of 0.0001. The neural network configuration was the same as described previously for the 8-bit odd parity characterization experiment above.

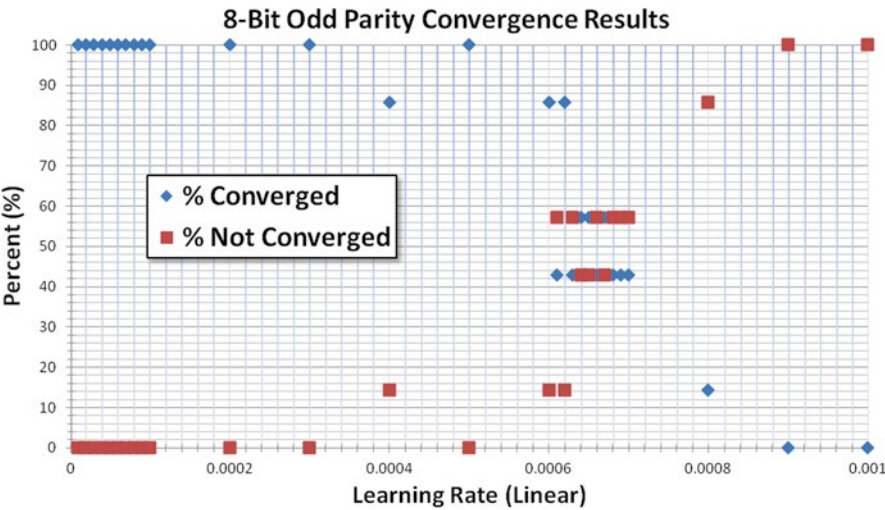


Fig. 2.11 8-Bit ODD parity function percentage convergence versus learning rate (in linear x-axis scale)

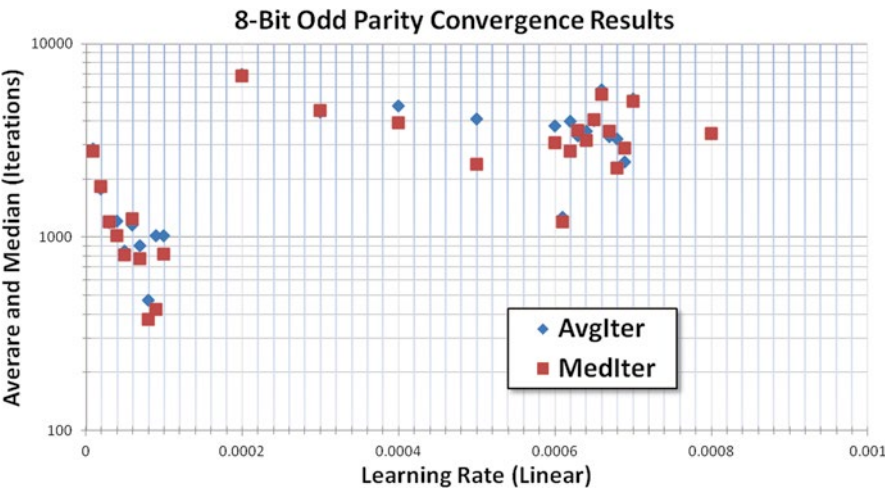


Fig. 2.12 8-Bit ODD parity function number of average (*diamond*) and median (*square*) iterations required for a successful convergence versus learning rate (in log x-axis scale)

In this case, the input training dataset was randomly selected during each epoch to contain 100, 95, 90, 80, 70, 60, 50, 40, 30, 20, 10, and 5 % of the complete dataset. For example, the last case scenario contained only 5 % of the original 256 record data set. The test phase contained the complete dataset and the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) rates were measured for each record. Figure 2.13 shows the results of this experiment. From the

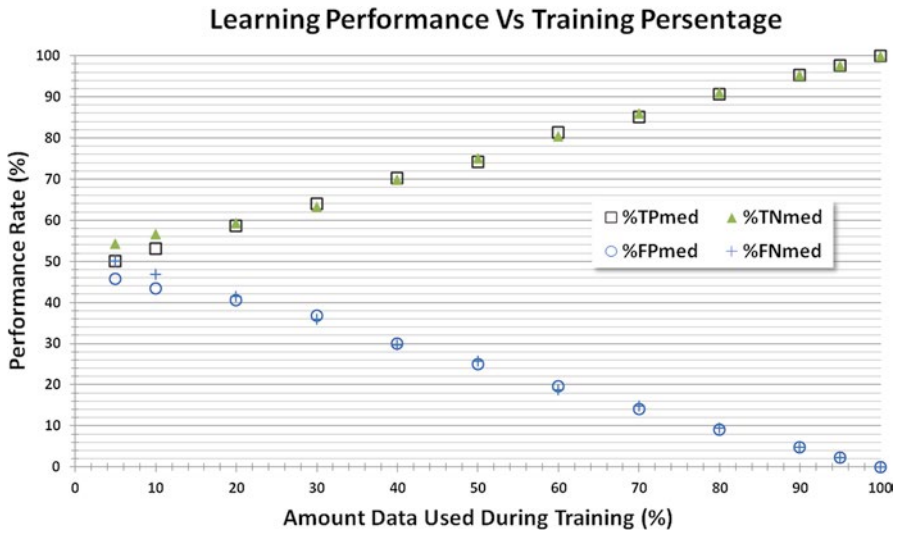


Fig. 2.13 Neural network performance as function of percent data used during training

figure, we can observe that as the percentage of training data is reduced the accuracy of the classification approaches 50 % which is that of mere chance. Above 90 % accurate classification was achieved for training data percentages of 80 % and above.

2.5 Brief Review of Neural Network Uses in Intrusion Detection Systems

In network security applications, neural networks have been commonly applied to addressing network intrusion detection problems. In 2006, Pervez and his research team [25] performed a comparative analysis of Artificial Neural Network (ANN) technologies applied to Intrusion Detection Systems (IDS). In their work, the authors focused on ANNs technologies designed to detect instances of the access of computer systems by unauthorized individuals and the misuse of system resources [25]. The authors described the work by Anderson [26] in which three threats were described which could be identified from audit data as: (1) External Penetrations—Unauthorized users of the system who gain access to the system; (2) Internal Penetrations—Authorized system users who utilize the system in an unauthorized manner; and (3) Misfeasors—Authorized user who misuse their access privileges. These threats represent areas in which a neural network could potentially be trained to identify patterns of malicious cyber behavior. Denning in 1987 provided some of the early work on intrusion detection systems which relied on identifying abnormal patterns of system usage [27]. Denning developed a model of a real-time intrusion-detection expert system capable of detecting break-ins, penetrations, and other forms of computer abuse. The model is based on the hypothesis that security

violations can be detected by monitoring a system's audit records for abnormal patterns of system usage. The model includes profiles for representing the behavior of subjects in terms of metrics and statistics, and rules for acquiring knowledge about this behavior from audit records and for detecting anomalous behavior. The model is independent of any particular system, application environment, system vulnerability, or type of intrusion, thereby providing a framework for a general-purpose intrusion detection expert system [27]. Overall, the model provides a framework for IDS research and development. For misuse detection, Cannady in 1998 [28] presented an approach utilizing the analytical strengths of neural networks and provided results indicating a data correlation rate of 97.56 %. In the experiments, the training of the neural network was performed using the Backpropagation algorithm for 10,000 iterations of selected training data for which 1,000 records were used for testing out of a total of 9,462 records available in their dataset. Their results demonstrated upwards of over 90 % correlation for various types of attacks that included SYNflood, SATAN, and ISS Scan attack tests [28]. Their results show the potential in using feed forward neural networks to identify specific attacks with a high degree of accuracy. The network traffic fields employed during training included:

- Protocol ID—The protocol associated with the event, (TCP=0, UDP=1, ICMP=2, and Unknown=3).
- Source Port—The port number of the source.
- Destination Port—The port number of the destination.
- Source Address—The IP address of the source.
- Destination Address—The IP address of the destination.
- ICMP Type—The type of the ICMP packet (Echo Request or Null).
- ICMP Code—The code field from the ICMP packet (None or Null).
- Raw Data Length—The length of the data in the packet.
- Raw Data—The data portion of the packet.

Another approach by Vollmer and Manic in 2009 involved researching computationally efficient implementations of neural networks in anomaly-based intrusion detection systems [29]. Their results maintained output accuracy while reducing by 70 % the amount of memory required to run the system. The neural network consisted of three fully connected layers and was trained and tested using ICMP rules and test data. One of the main advantages to employing neural networks in performing network security lies in the ability to store the various rule attack knowledge pattern vectors within the synaptic weights in between the neurons and layers [29] (graphically shown in Fig. 2.2) resulting in enhanced memory utilization and performance improvement.

2.6 Experimental Results and Discussion

It is clear artificial neural networks have been applied to various IDS challenges. However, it is difficult to compare any of the reported results given that most datasets employed are not publically available. Another point is whether those datasets

Table 2.2 Characteristics of the KDD dataset employed for training and characterization

Dataset Label	Total Attack	Total Normal	Total Records
Test data, corrected labels (for training)	250,436	60,593	311,029
Full data set (for test)	3,925,651	972,780	4,898,431

are relevant to DoD or military network operations. Therefore, we will compare our results on the classification of malicious flow network traffic to the results from a paper by Kayacik and team [30] that is based on a hierarchy of Self-Organizing Feature Maps. The goal in their work was to classify the KDD benchmark tagged dataset from the International Knowledge Discovery and Data Mining Tools Competition [31]. This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between “bad” connections, called intrusions or attacks, and “good” normal connections. This dataset contains a standard set of information to be audited, which includes a wide variety of intrusions simulated in a military network environment [31]. Overall, though this dataset is old, this is the most quoted dataset in the open published literature to the best of our knowledge. One interesting point of the work by Kayacik et al. is that they attempted to perform the classification using only 6 fields of the possible 41 features available in the dataset [30]. This goal resonates with our own in-house research goal of minimizing the form factor during IDS operations, which in this case was accomplished by reducing the total number of inputs to perform the classification. From their results, the detection rate on the test set varied between 89 and 99.7 % depending on the dataset partition employed [30].

To perform the neural network training with ALIEN, both the training and test data was used described in Table 2.2. The training dataset is composed of over 311,000 flow records, and the characterization, full data set contains close to 5 million flow records. For characterizing and measuring the performance of the ALIEN trained neural network only six fields were used and are described in Table 2.3. The description of the reminder of the 35 fields can be found on the KDD repository website [31].

During our experiments, in order to further minimize the size of our classifying neural network, we created a single layer perceptron neural network architecture as shown in Fig. 2.14. From the figure, we can observe that only six input neurons, six synaptic weights, and one output neuron were implemented. In addition, Table 2.4 shows an example training input vector from the KDD dataset that is mapped to the actual training input employed to the neural network. The example in Table 2.4 shows a traffic flow record corresponding to a warezmaster attack. In addition, the complete vector value mapping for the text fields 2, 3, and 4 in the KDD dataset and the desired neural network output (the last two columns on the right, traffic type label and neuron output) are shown in Table 2.5. In particular, the order input value from Table 2.5 corresponds to the input neural network value for fields 2, 3, and 4

Table 2.3 KDD fields used during training and performance characterization

• Field 1: <i>duration</i> , length (number of seconds) of the connection
• Field 2: <i>protocol_type</i> , type of the protocol, e.g. tcp, udp, etc.
• Field 3: <i>service</i> , on the destination, such as FTP, HTTP, Telnet, etc.
• Field 4: <i>flag</i> , normal or error status of the connection
• Field 5: <i>src_bytes</i> , number of data bytes from source to destination
• Field 6: <i>dst_bytes</i> , number of data bytes from destination to source

Fig. 2.14 Neural network experimental setup with a total of six input neurons, one output neuron, and six synaptic connections

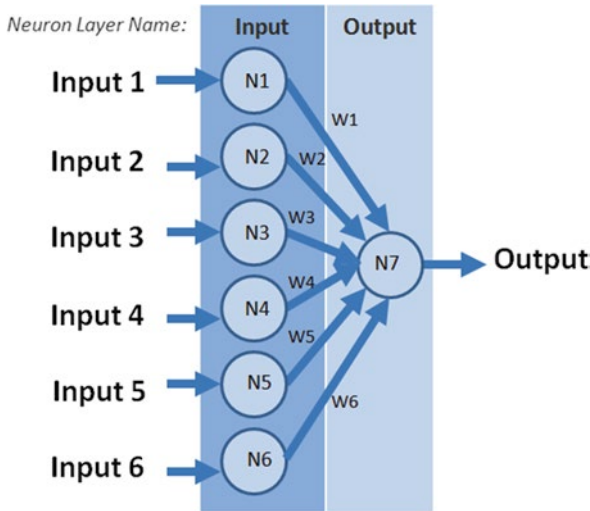


Table 2.4 Example of KDD fields used during training conversion to neural network input

KDD name	Duration	Protocol_type	Service	Flag	Src_bytes	Dst_bytes	Traffic type
Input	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Output
<i>Actual</i>	280	tcp	ftp_data	SF	283618	0	warezmaster
<i>Formatted</i>	280	2	17	10	283618	0	-1

of the KDD dataset. Also, the last two columns on the right of Table 2.5 correspond to the desired neural network output for the traffic_type label field where for all possible outputs only the normal output is set to 1, all the other possible traffic_type labels are set to -1 which correspond to the malicious network traffic. Given that the KDD training and characterization datasets are relatively large, 47.3 and 742.6 MB respectively, and given the limited computing performance of the desktop machine use to run the simulations, the training experiment was run for a maximum of three

Table 2.5 Input mapping where the order value represents the input to the neural network

Order	Protocol Type	Service	Flag	Order	Service	Order	Service	Traffic Type	Neural
Input	Field 2	Field 3	Field 4	Input	Field 3	Input	Field 3	Label	Output
1	icmp	auth	OTH	24	IRC	47	ije	back	-1
2	tcp	bgp	REJ	25	iso_tsap	48	shell	buffer_overflow	-1
3	udp	courier	RSTO	26	klogin	49	smtp	ftp_write	-1
4		csnet_ns	RSTOS0	27	kshell	50	sql_net	guess_passwd	-1
5		ctf	RSTR	28	ldap	51	ssh	imap	-1
6		daytime	S0	29	link	52	sunrpc	ipsweep	-1
7		discard	S1	30	login	53	supdup	land	-1
8		domain	S2	31	mtp	54	systat	loadmodule	-1
9		domain_u	S3	32	name	55	telnet	multihop	-1
10		echo	SF	33	netbios_dgm	56	tftp_u	neptune	-1
11		eco_i	SH	34	netbios_ns	57	time	nmap	-1
12		ecr_i		35	netbios_ssn	58	tim_i	normal	1
13		efs		36	netstat	59	urp_i	perl	-1
14		ex ec		37	nmsp	60	uucp	phf	-1
15		finger		38	nntp	61	uucp_path	pod	-1
16		ftp		39	ntp_u	62	vmnet	portsweep	-1
17		ftp_data		40	other	63	whois	rootkit	-1
18		gopher		41	pm_dump	64	X11	satant	-1
19		hostnames		42	pop_2	65	Z39_50	smurf	-1
20		http		43	pop_3			spy	-1
21		http_443		44	printer			teardrop	-1
22		icmp		45	private			warezclient	-1
23		imap4		46	remote_job			warezmaster.	-1

to five iterations and one epoch each time. Typical results after training the neural network with ALIEN for 5 iterations and 1 epoch showed a detection rate between 96.07 and 98.46 %. These results are very encouraging for such simple neural network architecture.

The second classification experiment performed was to train a feed forward neural network using ALIEN to perform classification between DNS A and MX network traffic record requests. An MX record or mail exchanger record is a type of record in the Domain Name System (DNS) that specifies a mail server that is responsible for accepting email exchanges for the particular domain of the recipient. The A type address record returns the 32-bit IPv4 address, and it is commonly used to map hostnames to the IP address of the host.

The process to capture an MX or A record can be accomplished by using `tcpdump`, which is commonly used open source command line software for packet capture and analysis [33]. The specific command that is used to capture a single network packet can be written as:

```
$ sudo tcpdump -i any -nnvXA -c 1 dst port 53 > filename
```

The commands to request an MX or A packet from the network can be written, for example, as:

```
$ dig www.google.com mx
$ dig www.google.com a
```

In the command lines above, A and MX DNS records were requested from the www.google.com domain name and the results are shown in the figures below for each DNS record request respectively.

Figure 2.15 displays a typical `tcpdump` output example of an MX network packet as was used to perform our training. Figure 2.16 displays a typical `tcpdump` output example of an A network packet that was used to perform our training and classification experiment. Per RFC 1035 [32], the location that tells us whether the record represents an A or MX DNS request follows the QNAME byte sequence describing the queried domain name (www.google.com in the above example). For our specific training experiment, the packets were truncated to a length of 125 bytes containing only the hexadecimal information as shown in Figs. 2.17 and 2.18 for each of the MX and A DNS requests. For any truncated packet that was shorter than 125 bytes, null bytes (0x00) were used to pad the missing length to 125 bytes as displayed in the figures. In addition, each packet was labeled as indicated in Figs. 2.17 and 2.18 to identify the request as either an MX lookup or an A lookup.

Finally, each byte was converted to unsigned integer in the inclusive range between 0 and 255. The request type label of each packet was then represented as -1 for MX or 1 for A as shown in Figs. 2.19 and 2.20. The training records as actually submitted to the neural network are exemplified by Figs. 2.19 and 2.20.

During the MX and A DNS record classification experiment, 5,000 A and 32 MX DNS record request packets were captured employing the `tcpdump` software application respectively. In summary, ALIEN was able to successfully train a 63 input, 10 hidden, and 1 output neuron network configuration at a learning rate of 0.1 within

```

23:37:07.076229 IP (tos 0x0, ttl 64, id 31915, offset 0, flags [none], proto
UDP (17), length 71)
127.0.0.1.51324 > 127.0.1.1.53: 56852+ [1au] MX? www.google.com. (43)
0x0000: 4500 0047 7cab 0000 4011 fef8 7f00 0001 E..G|...@.....
0x0010: 7f00 0101 c87c 0035 0033 ff46 de14 0120 .....|.5.3.F....
0x0020: 0001 0000 0000 0001 0377 7777 0667 6f6f .....www.goo
0x0030: 676c 6503 636f 6d00 000f 0001 0000 2910 gle.com.....).
0x0040: 0000 0000 0000 00 .....

```

Fig. 2.15 MX DNS network packet captured with tcpdump, the byte containing the record type is *underlined* and in *boldface*

```

23:34:53.162407 IP (tos 0x0, ttl 64, id 31914, offset 0, flags [none], proto
UDP (17), length 71)
127.0.0.1.55133 > 127.0.1.1.53: 53846+ [1au] A? www.google.com. (43)
0x0000: 4500 0047 7caa 0000 4011 fef9 7f00 0001 E..G|...@.....
0x0010: 7f00 0101 d75d 0035 0033 ff46 d256 0120 .....|.5.3.F.V..
0x0020: 0001 0000 0000 0001 0377 7777 0667 6f6f .....www.goo
0x0030: 676c 6503 636f 6d00 0001 0001 0000 2910 gle.com.....).
0x0040: 0000 0000 0000 00 .....

```

Fig. 2.16 A DNS network packet captured with tcpdump, the byte containing the record type is *underlined* and in *boldface*

```

450000477cab00004011fef87f0000017f000101c8
7c00350033ff46de140120000100000000000010377
777706676f6f676c6503636f6d00000f0001000029 MX?

```

Fig. 2.17 Truncated hex representation of an MX DNS lookup, the byte containing the record type is *underlined* and in *boldface*

```

450000477caa00004011fef97f0000017f000101d7
5d00350033ff46d2560120000100000000000010377
777706676f6f676c6503636f6d0000010001000029 A?

```

Fig. 2.18 Truncated hex representation of an A DNS lookup, the byte containing the record type is *underlined* and in *boldface*

```

69,0,0,71,124,171,0,0,64,17,254,248,127,0,0,1,127,0,1,1,200,
124,0,53,0,51,255,70,222,20,1,32,0,1,0,0,0,0,0,1,3,119,119,
119,6,103,111,111,103,108,101,3,99,111,109,0,0,15,0,1,0,0,0,41,1

```

Fig. 2.19 Neural network input for MX network packet, the byte containing the record type is *underlined* and in *boldface*

```

69,0,0,71,124,170,0,0,64,17,254,249,127,0,0,1,127,0,1,1,215,
93,0,53,0,51,255,70,210,86,1,32,0,1,0,0,0,0,0,1,3,119,119,
119,6,103,111,111,103,108,101,3,99,111,109,0,0,1,0,1,0,0,0,41,-1

```

Fig. 2.20 Neural network input for A network packet, the byte containing the record type is *underlined* and in *boldface*

approximately 30 iterations. ALIEN’s classification was perfect in all but two MX records where the telltale byte identifying the lookup type was outside the truncation window. This means that for variable domain names, neural networks are not suitable for classification as the number of inputs must remain the same. A possible solution is to have very long inputs up to the maximum possible number of bytes; however, the performance of the neural network would be decrease as the computing resources would increase exponentially. Still the overall classification rate was 99.96 % with only the two MX packets not achieving proper classification as the required information was missing from the truncated record.

After having achieved success with ALIEN and learning its limitations with variable length domain names, we designed an experiment in which to break down the problem intelligently to allow feature extraction to clearly highlight the problem in question. For example, in our DNS A and MX record request classification problem, the challenge to solve is simply to find and correctly classify two specific requests. Therefore, the problem should be easy for a neural network to solve as the choices are only limited two possible solutions, A or MX. In the updated approach to feature extraction, the neural network inputs are the respective MD5 digests of the DNS question QNAME, QTYPE, and QCLASS fields normalized to a decimal representation between -1.0 and 1.0 . Using a one-way hashing function digest to represent the QNAME, QTYPE, and QCLASS fields allows the classifier to accept these fields as distinct features free from dependency upon the length or byte-offset position of each field. Since there is no cryptographic strength requirement for this particular hashing application, MD5 was chosen as an expedient option because an implementation already exists within the “hashlib” package of the standard CPython 2.7.3 distribution. Figs. 2.21 and 2.22 describe how the question section of the DNS request packet was divided into three distinct inputs that cover the domain name (www.google.com, used as an example), query type (A and MX), and the query class of the lookup being performed.

The experimental results of the training performed using the new methodology is displayed in Figs. 2.23 and 2.24.

Question section of DNS A lookup	<code>\x03www\x06google\x03com\x00\x00\x01\x00\x01</code>
Input vector divided into three parts	<code>\x03www\x06google\x03com\x00 , \x00\x01 , \x00\x01</code>
Hashed inputs to Neural Network	<code>-0.521159634814 , 0.0901948225429 , 0.0901948225429 , -1</code>

Fig. 2.21 DNS A network packet and conversion to neural network input

Question section of DNS MX lookup	<code>\x03www\x06google\x03com\x00\x00\x0f\x00\x01</code>
Packet divided into three inputs	<code>\x03www\x06google\x03com\x00 , \x00\x0f , \x00\x01</code>
Hashed inputs to Neural Network	<code>-0.521159634814 , -0.434861249159 , 0.0901948225429 , 1</code>

Fig. 2.22 DNS MX network packet and conversion to neural network input

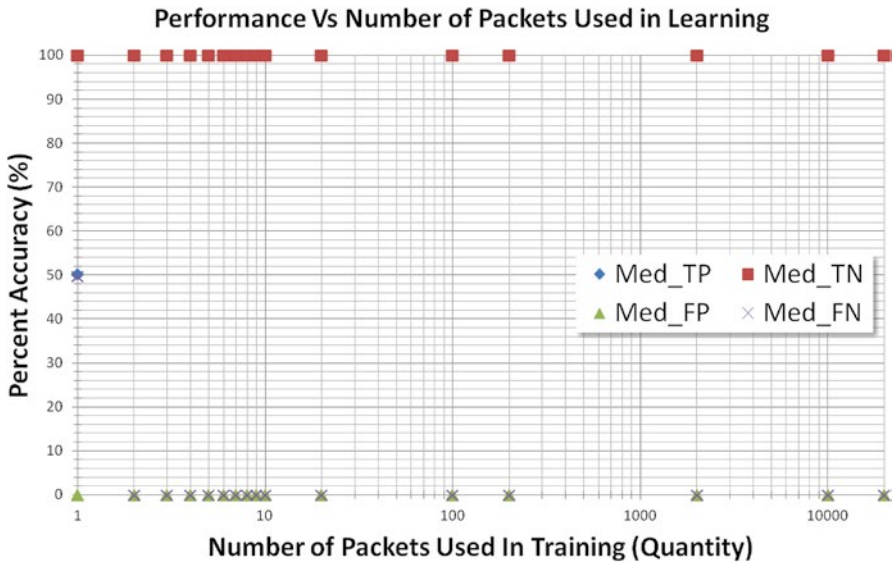


Fig. 2.23 ALIEN Neural network learning classification performance as function of number of number of actual packets used during training

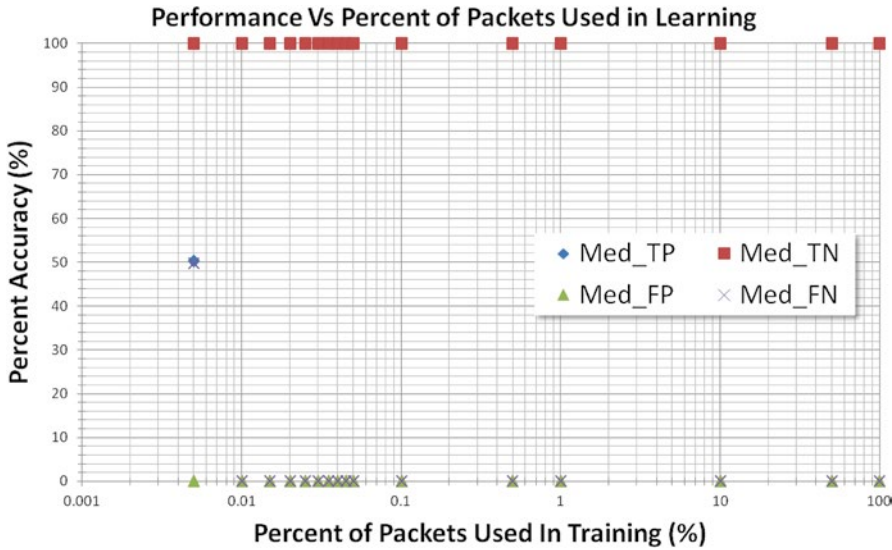


Fig. 2.24 ALIEN Neural network learning classification performance as function of number of number of percent data used during training

Figure 2.23 shows the percent accuracy versus the number of packets used during training. From this Figure, we can observe that if only two packets are used for ALIEN to train the neural network, the classification performance is 100 %. It is interesting to note that if only one packet is used, the true-negative, TN, performance

is 100 % while the true-positive, TP, performance is about 50 %. What this specific result highlights is the fact that only one packet of the MX or A type was used during the training. Therefore, the system learned to identify DNS A packet types and guessed with 50 % accuracy on the DNS MX packet type. Figure 2.24 shows the same results of classification accuracy versus the percentage of data used to perform the training of the neural network using ALIEN. The results and figures clearly highlight that given the lowest possible amount of data, 2 packets, the network is able to produce perfect classification results with 100 % accuracy for a total of 20,000 distinct network packets perfectly classified for domain names varying from 5 to 30 bytes in length.

During the previous experiments, the training dataset was randomly chosen. Therefore, it is possible that in some cases, the number of A versus MX packets used during training was greater than the other or vice versa. This would mean the training dataset was not balanced in terms of the representation of the possible lookup types. Thus, we performed an additional experiment in which we ensured that the random dataset selection process included an equal number of A and MX DNS packets. Figure 2.25 shows the experimental results for classifying A and MX packets as function of number of packets used during training. During the experiment, the number of MX and A packets was balanced but the selection of each was random. We performed 100 epochs for each number of packets, and the results reinforce our previous conclusion that ALIEN can achieve 100 % perfect accuracy of classification amongst the 20,000 network packets used during our experiments.

Figures 2.26 and 2.27 display the number of iterations required to achieve perfect classification as function of percent data used and number of packets used. From the

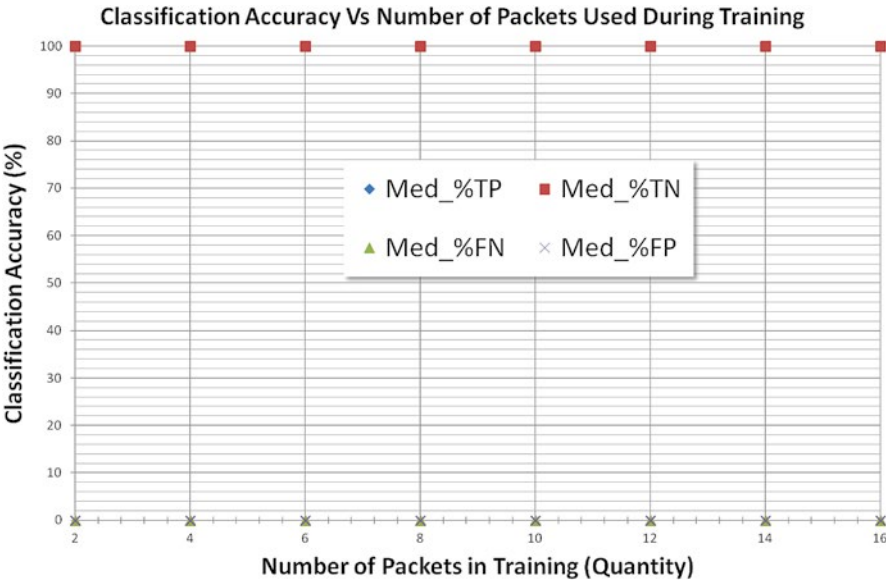


Fig. 2.25 ALIEN Neural network learning classification performance as function of number of number of packets used during training while ensuring that a balanced training dataset was used

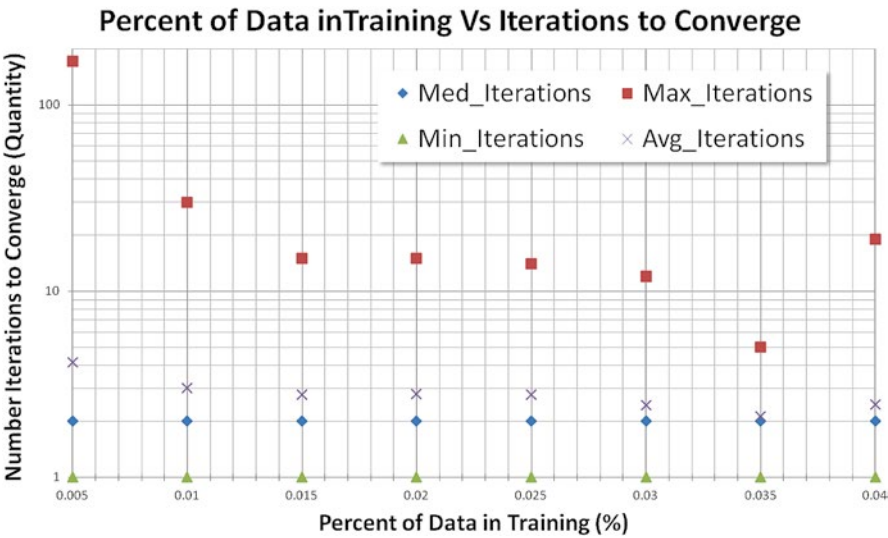


Fig. 2.26 ALIEN neural network learning classification performance for the number of iterations required for perfect classification as function of percent data used during training while ensuring that a balanced training dataset was used

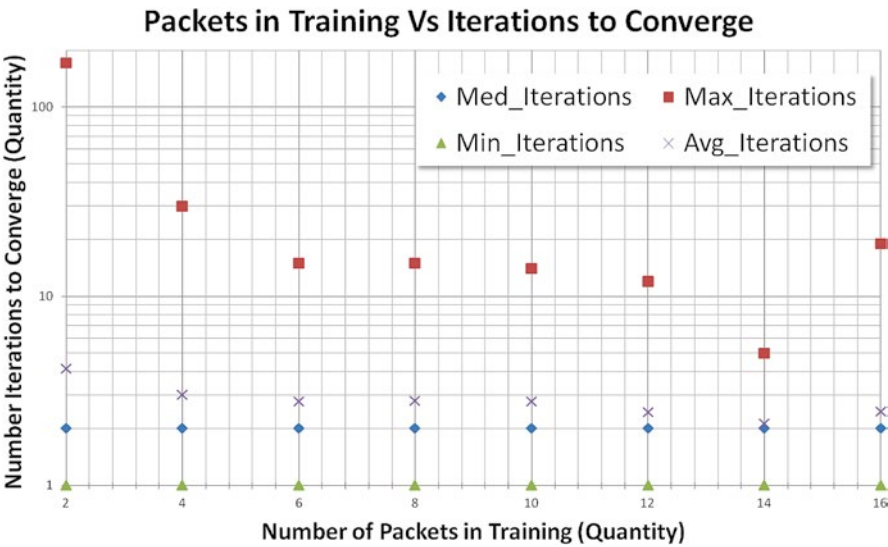


Fig. 2.27 ALIEN neural network learning classification performance for the number of iterations required for perfect classification as a function of number of packets used during training while ensuring that a balanced training dataset was used

results, we can observe that the minimum number of iterations achieved was one single iteration and the maximum covered 100 iterations. In general, the median number of iterations required remained two iterations for all cases during our experiments.

2.7 Conclusions

In this work, we presented a newly developed neural network learning algorithm called ALIEN. The goal of the work is to develop a low architectural overhead learning algorithm to train a memristor-based physical neural network for applications in network security in the tactical environment. This work demonstrated ALIEN's capacity to perform nonlinear classification in the performance of pattern classification. For instance, our experiments yielded 98.46 % detection rate for the KDD99 data set. And in the classification of A and MX DNS record requests, our experiments yielded near perfect results except where the critical information required for classification was missing within the truncated record. This result highlighted a weakness in the ability for a fixed input neural network to be able to effectively classify variable domain name record requests. To solve this challenge, we brokered the problem by dividing DNS QNAME, QTYPE, and QCLASS into three distinct inputs that were easily classified by the neural network. It is evident that this last step trivialized the problem to a three input simple challenge. During this particular version of the experiment, we were able to demonstrate 100 % accuracy classifying 20,000 DNS A and MX records when only two packets were used during training (using one of each A and MX record type).

2.8 Future Work

A potential next step could include porting ALIEN to an FPGA computing architecture where we can boost the computing performance and be able to process an increased number of inputs. We would like to be able to process over 1,000 distinct inputs that would allow for broader classification potential. This will offer the opportunity to test and validate neural network-based network security solutions within an embedded sensor architecture. In addition, we would like to enhance the capabilities of the ALIEN platform to include the ability for the neuron outputs to be analog. This will give us the ability to measure level of confidence in the classification by each neuron and mimic closely the operation of memristor devices within the neural network and characterize the capabilities and limitations of the technology. The idea is for ALIEN to reflect the analog output performance of memristor-based physical neural networks. Also, we would like to evaluate ALIEN's applicability to more practical scenarios such as hierarchical network traffic analysis. The problems may range in granularity from the very specific (such as segregating different DNS

request types from amongst each other beyond A and MX DNS records as shown here) to the very general (such as classifying an arbitrary network traffic sample as normal or anomalous based upon an established baseline, similar to the KDD99 challenge). Our hope is that this work will pave the way for the eventual implementation of powerful neuromorphic intrusion detection/prevention solutions in software within a high performance computing cluster formed by multi-core processors, GPUs, or FPGAs and in the future within embedded memristive-based hardware platforms that are small, light, and low in power consumption.

References

1. John Gantz and David Reinsel, "THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East", IDC IVIEW Report, <http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf>, Sponsored by EMC. Dec. 2012.
2. L.O. Chua, "Memristor - the missing circuit element," IEEE Trans. Circuit Theory, 18 (1971) 507–519.
3. Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams, "The missing memristor found," Nature 453 (2008), 80–83.
4. R. Williams, "How We Found The Missing Memristor," IEEE Spectrum, 45 (2008) 28–35. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4687366&isnumber=4467055>
5. R.E Pino, J.W. Bohl, N. McDonald, B. Wysocki, P. Rozwood, K. Campbell, A Timilsina, "Compact method for modeling and simulation of memristor devices: Ion conductor chalcogenide-based memristor devices," 2010 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Anaheim, CA, June 17–18 (2010) pp 1–4. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5510936&isnumber=5510921>
6. Chris Yakopcic, Tarek M. Taha, Guru Subramanyam, and Robinson E. Pino, "Memristor SPICE Model and Crossbar Simulation Based on Devices with Nanosecond Switching Time," Proceedings of International Joint Conference on Neural Networks, Dallas, Texas, USA, August 4–9, 2013, pp.
7. C. Yakopcic, T.M. Taha, G. Subramanyam, R.E. Pino, S. Rogers, "A Memristor Device Model," IEEE Electron Device Letters, 32 (2011) 1436–1438.
8. Robert Kozma, Robinson E. Pino, and Giovanni E. Paziienza, "Are Memristors the Future of AI?," R. Kozma et al. (eds.), Advances in Neuromorphic Memristor Science and Applications, Springer, New York, 2012, pp 9–14.
9. Qiangfei Xia, Warren Robinett, Michael W. Cumbie, Neel Banerjee, Thomas J. Cardinali, J. Joshua Yang, Wei Wu, Xuema Li, William M. Tong, Dmitri B. Strukov, Gregory S. Snider, Gilberto Medeiros-Ribeiro, and R. Stanley Williams, "Memristor–CMOS Hybrid Integrated Circuits for Reconfigurable Logic," Nano Letters 9 (2009) 3640–3645.
10. M. Shevenell, "Task 2 - Research Emerging Technologies for Embedded High Performance Intelligent Sensor," ARL internal presentation, Security for Tactical Operations Relying on Methods for Enhancing Robustness (STORMER), 1st Quarter Review 2013 STORMER Team,
11. Joshua S. White, Thomas Fitsimmons, Jeanna N. Matthews, "Quantitative Analysis of Intrusion Detection Systems Snort and Suricata," SPIE Defence and Security, Sensing, Cyber Sensing 2013, Proc. of SPIE 8757 (2013) pp. 875704–1–12.
12. PowerEdge R717 Technical Guide, <http://www.dell.com/downloads/global/products/pedge/en/Poweredge-r715-technicalguide.pdf> (Viewed on September 12, 2013).
13. FAQs, Raspberry Pi, <<http://www.raspberrypi.org/faqs>>, (9 September 2013)

14. R. Pino, "Reconfigurable Electronic Circuit," United States Patent 7902857, March 8 (2011).
15. R. Pino, "Neuromorphic Computer," United States Patent 8275728, September 25 (2012).
16. R. Pino, J. Bohl, "Self-Reconfigurable Memristor-Based Analog Resonant Computer," United States Patent 8274312, September 25 (2012).
17. Robinson E. Pino, Youngok K. Pino, "Reconfigurable memristor-based computing logic," United States Patent US 8427203 B2, April 23 (2013).
18. G.S. Rose, R. Pino, Q. Wu, "A low-power memristive neuromorphic circuit utilizing a global/local training mechanism," Proceedings of International Joint Conference on Neural Networks (IJCNN), San Jose, CA, July 31-5, 2011, pp. 2080 – 2086.
19. David Bol, Dina Kamel, Denis Flandre, Jean-Didier Legat, "Nanometer MOSFET effects on the minimum-energy point of 45nm subthreshold logic," Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design (ISLPED), San Francisco, CA, August 19–21, 2009, pp. 3-8.
20. R. Pino, H. Li, Y. Chen, M. Hu. B. Liu, "Statistical memristor modeling and case study in neuromorphic computing," 2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, June 3-7, 585 - 590 (2012).
21. McCulloch and Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, 5 (1943) 115-133.
22. O. L. Mangasarian, "Linear and Nonlinear Separation of Patterns by Linear Programming," *Operations Research*, 13 (1965) 444-452.
23. Paul J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," PhD thesis, Harvard University (1974).
24. Paul J. Werbos, "Backpropagation through time: what it does and how to do it," Proceedings of the IEEE, 78 (1990) 1550 – 1560.
25. S. Pervez, I. Ahmad, A. Akram, S. U. Swati, "A Comparative Analysis of Artificial Neural Network Technologies in Intrusion Detections Systems," Proc. of the 6th WSEAS International Conference on Multimedia, Internet & Video Technologies, Lisbon, Portugal, September 22-24 (2006).
26. J.P. Anderson, "Computer Security Threat Monitoring and Surveillance," Technical Report, J.P. Anderson Company, Fort Washington, Pennsylvania April (1980).
27. D. Denning, "An Intrusion-Detection Model," IEEE Tran Software Engineering, SE-13(2) 222-232 (1987).
28. J. Cannady, "Artificial Neural Networks for Misuse Detection," Proc. 21st National Information Systems Security Conference, Arlington, VA, October 5-8 (1998).
29. T. Vollmer, M. Manic, "Computationally Efficient Neural Network Intrusion Security Awareness," Proc. 2nd International Symposium on Resilient Control Systems, Idaho Falls, ID, August 11 - 13 (2009).
30. H.G. Kayacik, A.N. Zincir-Heywood, M.I. Heywood, "On the capability of an SOM based intrusion detection system," 2003 IEEE Proceedings of the International Joint Conference on Neural Networks (IJCNN 2003) July 20-24, 2003, pp.1808,1813
31. KDD Cup 1999 Data, The UCI KDD Archive Information and Computer Science University of California, <<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>> (9 September 2013).
32. Domain Names - Implementation And Specification, <http://www.ietf.org/rfc/rfc1035.txt>, (viewed on September 11, 2013)
33. tcpdump, <http://www.tcpdump.org/> (viewed on July 23, 2013).

Cybersecurity Systems for Human Cognition
Augmentation

Pino, R.E.; Kott, A.; Shevenell, M. (Eds.)

2014, XVI, 209 p. 113 illus., Hardcover

ISBN: 978-3-319-10373-0