4-26-2020

Machine Learning for Cyberattack Detection

Kayla Chisholm
University of Dayton

Machine Learning for Cyberattack Detection

Honors Thesis

Kayla Chisholm

Department: Electrical and Computer Engineering

Advisor: Chris Yakopcic, Ph.D.

April 2020

Machine Learning for Cyberattack Detection

Honors Thesis

Kayla Chisholm

Department: Electrical and Computer Engineering

Advisor: Chris Yakopcic, Ph.D.

April 2020

Abstract

Machine learning has become rapidly utilized in cybersecurity, rising from almost non-existent to currently over half of cybersecurity techniques utilized commercially. Machine learning is advancing at a rapid rate, and the application of new learning techniques to cybersecurity have not been investigate yet. Current technology trends have led to an abundance of household items containing microprocessors all connected within a private network. Thus, network intrusion detection is essential for keeping these networks secure. However, network intrusion detection can be extremely taxing on battery operated devices. The presented work presents a cyberattack detection system based on a multilayer perceptron neural network algorithm. To show that this system can operate at low power, the algorithm was executed on two commercially available minicomputer systems including the Raspberry PI 3 and the Asus Tinkerboard. An analysis of accuracy, power, energy, and timing was performed to study the tradeoffs necessary when executing these algorithms at low power. Our results show that these low power implementations are feasible, and a scan rate of more than 226,000 packets per second can be achieved from a system that requires approximately 5W to operate with greater than 99% accuracy.

Table of Contents

List of Figures

List of Tables

Introduction

Computer networks have steadily increased in size and complexity since their inception. As computer networks grow, it becomes more difficult to provide reliable network security [1], especially in low power portable systems. In current commercial technology, many electronic devices and systems possess both a microprocessor and a connection to a network. Unless each of these devices are equipped with up to date network security, users are exposing their networks to a substantial number of access points. Therefore, high speed energy efficient intrusion detection must be utilized to protect systems from the rapidly evolving cyberattack landscape.

Intrusion detection systems that use neural networks and machine learning can be utilized as an efficient alternative to traditional security systems. Neural networks are extremely capable in the fields of image and pattern recognition [2,3]. These capabilities can be leveraged in intrusion detection applications as the networks are trained to recognize the difference between benign data and a cyberattack. Certain neural network algorithms have adaptive properties [4], thus they can self-optimize during normal operation. The ability to improve during while in use would be useful in an intrusion detection system because new cyberattacks are constantly being developed. Therefore, it would be possible for an adaptive system to catch new attacks without undergoing a costly retraining process. Many neural network algorithms are implemented using layers of vector-matrix multiplication [5]. This type of algorithm can be implemented in an extremely parallel design using specialized low power hardware [5-10]. Thus, neural network based intrusion detection could be performed at extreme low power.

Before low power, energy efficient machine learning based intrusion detection systems are produced, different learning algorithms need to be examined to determine which are most appropriate for recognizing attacks within network data. Thus, this work presents a comparison of two perceptron topologies running on two different low power hardware systems. For training and testing this work uses the NSL-KDD dataset [11], an update to the Knowledge Discovery and Datamining dataset [12]. In this work we train two different multilayer perceptron algorithms, one with a single hidden layer, and one with two hidden layers. In addition to studying perceptron algorithm accuracy [1], in this

work we study low power implementations of these algorithms on a Raspberry Pi 3 [13] and an Asus Tinkerboard [14]. These minicomputers are low power (<6W), handheld (<60g) computing systems with a high degree of flexibility. Our results present a design space analysis that discusses the tradeoff of using these systems in terms of accuracy, power, energy, and time. By using these systems, we show that network intrusion detection can be implemented on lower power systems with greater than 99% accuracy with a scanning rate of more than 226,000 packets / second.

Related work in this area shows similar studies where neural network algorithms are used to carry out intrusion detection using similar training and testing datasets. Both convolutional [15] and deep learning [16] architectures have been applied to this problem, and these approaches achieve high accuracy, which is about 99% in some cases. Hebbian learning has also been applied to the cyber-attack detection problem, and significant detection improvement is observed when using a multiscale learning rule [17]. In our previous work, we compare perceptron algorithms that differ in size and complexity in terms of attack detection accuracy [18]. Furthermore, our research group has studied the implementation of network intrusion detection on more exotic hardware architectures including the IBM True North spiking processor [19], as well as simulated memristor hardware [20]. However, to the best of our knowledge, no published work has carried out a power, energy, and timing comparison when moving these algorithms to low size, weight, and power commercial off the shelf systems. The presented work demonstrates the possibility efficiently implementing intrusion detection using low cost components.

This work is organized as follows: Section II details the necessary background to understand the objectives of this work, Section III describes neural network algorithms, Section IV discusses the perceptron algorithms used in this work, and Section V describes the NSL-KDD dataset used in this work. Section VI shows the results obtained when training and testing one of the network topologies using MATLAB. Section VII discusses the results found when training and testing the network topologies using TensorFlow and low power hardware. Section VIII discusses the hardware evaluation results that compare these networks in terms of power, energy, and time. Finally, Section IX provides a brief conclusion.

Background

Prior to implementing neural network layouts on a working knowledge of machine learning, neural networks, and intrusion detection was developed. Traditional network intrusion detection systems are general rule based (such as Snort). Because traditional systems follow rules based on attacks that were previously determined to be malicious, they are generally unable to recognize new attacks. New malicious attacks are continuously created. Given the goal of our work is to develop a low power efficient intrusion detection system, the traditional system would not be the most effective given that it generally is not good at recognized new attacks. Therefore, our approach to network intrusion detection is a neural network pattern based system that can potentially recognize patterns among malicious attacks in order to identify them.

Neural networks have various practical applications including facial recognition, voice recognition, and medical image recognition. Ultimately, the task was to develop low power machine learning algorithms that can serve as efficient intrusion detection systems given the power constants of low power systems. In order to develop the low power intrusion detection system, we studied neural networks to determine which were the most promising for detecting cyberattacks, we tested the neural networks for efficiency using a portable computer, we then tested the neural networks for efficiency using low power hardware.

Neural Network Algorithms

Before determining which neural network algorithm to implement and test on low power systems, a survey of previous studies using neural networks was completed. Table I displays the previous work using neural networks for cyberattack detection and the corresponding accuracies as reported in the studies. As shown in Table I, Multilayer Perceptron Algorithms consistently have accuracies higher than 99% [15, 17, 18, 19].

Table I. Accuracies for Various Neural Networks Studied in Previous Work.

Network Algorithm	Accuracy (%)
Multiscale Hebbian	93.56
Deep Convolutional Network	98.83
Multilayer Perceptron – 1 hidden layer	99.85
Multilayer Perceptron – 2 hidden layers	99.68
Multilayer Perceptron – 3 hidden layers	99.78
Convolutional Neural Network – 1 layer	99.9
Convolutional Neural Network – 2 layers	99.8
Convolutional Neural Network – 3 layers	80.1
Deep Feed Forward – 1 layer	92.9
Deep Feed Forward – 2 layers	92.9
Deep Feed Forward – 3 layers	93.0
Deep Feed Forward – 4 layers	93.0
Deep Feed Forward – 5 layers	92.9

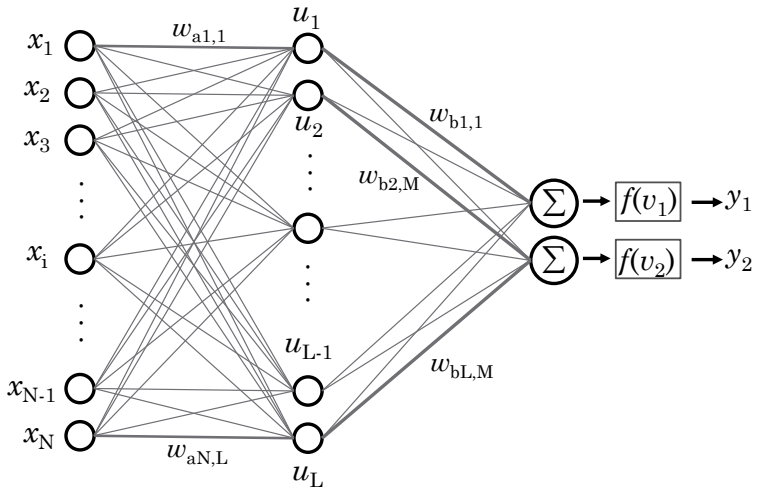
Key differences between the work depicted in Table 1 and the work that we have done is that we will be testing the algorithms using low power systems.

Perceptron Learning Algorithms

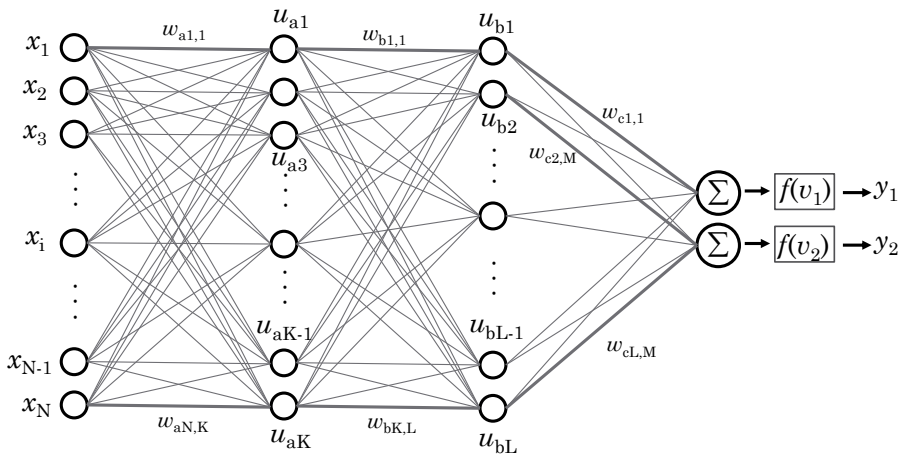
A neural network is a computing system inspired by human brain function. Neural networks "learn" to complete tasks by recognizing patterns rather than following a set of rules. They generally operate in two stages: training and testing. During the training stage, the system is shown data which it then classifies. After the network makes a classification determination, an error calculation is determined. Depending on how accurate the system's classifications were, weights will be updated accordingly. This process will continue in stages until the system processes the data for a predetermined number of stages. Once the process is complete the weights remain unchanged and the system moves on to the testing stage. During the testing stage, the system is again shown data to classify. After each data packet is classified, a final accuracy is determined. This type of system could allow for real-time updates and feedback and could also prevent new attacks.

$$v_j = \sum_{i=1}^{N} x_i w_{ij} \quad (1)$$

$$y_j = \frac{1}{1+e^{-v}} \quad (2)$$



(a)



(b)

Figure 1. Two different multilayer perceptron topologies presented in this work including (a) a topology with one hidden layer and (b) a topology with two hidden layers.

After surveying various types of neural network algorithms and other network intrusion detection studies [19], the multilayer perceptron was determined to be the first neural network type to examine on both a computer system and low power hardware. Two layouts of the multilayer perceptron were tests, one with a single hidden layer and one with two hidden layers (see Fig. 1). Perceptron algorithms possess two different modes of operation, as discussed above, testing and training. During the training phase, an iterative algorithm updates the weights w with the goal of minimizing error over time. Once training is finished, the resulting network should provide strong classification accuracy when presented with data like the training data. In this work, we utilize the standard back-propagation training algorithm [22]. Corresponding to Fig. 1, the intermediate output…

The multilayer perceptron in Figure 1 (a) contains one layer of neurons between weight matrices w_a and w_b. The input pattern must traverse each of these layers before a classification result is produced. Increasing the number of neurons layers in a perceptron typically produces a system with a higher classification accuracy. With more layers, more complex distinctions can be made as flexibility in feature classification is increased. Fig. 1 (b) shows multilayer perceptron with two hidden neuron layers. In the following section, we analyze the differences between these two network layouts in terms of accuracy, energy, and timing.

Cyberattack Dataset

To train and evaluate the intrusion detection networks, we make use of an improvement to the Knowledge Discovery and Datamining (KDD) data set which is known to make the data more appropriate for examining the effectiveness of artificial neural networks. This dataset is known as NSL-KDD [11]. NSL-KDD removes all repeated identical data points from the KDD dataset [12] and provides more balance to the different classes within the data.

More specifically, for the experiments in this work, the entirety of the data found in the "KDDtrain+.txt" file was used and can be found at [11]. This dataset contains several different types of cyberattacks and are detailed in Table II. Twenty-two different attacks are present in this data, and a large amount of benign data is also present in this data, and a large amount of benign data is also present. The systems presented in this work must learn to distinguish between the normal data and the attack data with the highest accuracy possible.

The differences between a normal packed (see Figure 2 (a)) and an attack packet (see Figure 2 (b)) in the NSL-KDD dataset are not easily recognizable, but patterns in the features are likely to emerge when scanning a large number of samples. To take advantage of each of the features in this data during training and classification, some minor preprocessing was performed. Each packet contains 43 values, and the 42nd value (displayed as either normal or Neptune in Figure 2) was sued as a label during training. The 43rd entry is a number related to classification difficulty which was removed and not used in this experiment. Additionally, the twentieth entry has a value of 0 for all packets in the dataset, so this value was removed from all packets since it would not affect training accuracy. The features (displayed as tcp, http, and SF in Figure 2 (a)) were non-numerical, so they were converted to integer values. For example, the fourth feature from the left 9SF in Figure 2 (a)) is a label capable of storing one of eleven possible strings, thus the string contained in this feature is now represented by a number 1 through 11 (depending on the string present). The data was also normalized (across samples) so that the largest value in any feature column was 1. Figure 3 shows the same example packets in Figure 2 after processing has been applied.

Table II. Breakdown of Different Attack Types within the NSL-KDD Dataset.

Attack Type	Number in Dataset	Attack Class
back	956	DOS
buffer_overflow	30	U2R
ftp_write	8	R2L
guess_passwd	53	R2L
imap	11	R2L
ipsweep	3599	PROBE
land	18	DOS
loadmodule	9	U2R
multihop	7	R2L
neptune	41214	DOS
nmap	1493	PROBE
normal	67343	NORMAL
perl	3	U2R
phf	4	R2L
pod	201	DOS
portsweep	2931	PROBE
rootkit	10	U2R
satan	3633	PROBE
smurf	2646	DOS
spy	2	R2L
teardrop	892	DOS
warezclient	890	R2L
warezmaster	20	R2L

```
0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,150,25,0.17,
0.03,0.17,0.00,0.00,0.00,0.05,0.00,normal,20
```

(a)

```
0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
123,6,1.00,1.00,0.00,0.00,0.05,0.07,0.00,255,26,0.10,
0.05,0.00,0.00,1.00,1.00,0.00,0.00,neptune,19
```

(b)

Figure 2. Examples of packets contained in the NSL-KDD dataset displaying (a) a normal packet, and (b) an attack.

12	←	"normal" was removed from the pattern and correlated to an attack type category number (that can be any number 1 to 23).

```
0,0.6667,0.300,0.9091,3.5581e-07,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0.0039,0.0039,0,0,0,0,1,0,0,0.5882,
0.0980,0.1700,0.0300,0.1700,0,0,0,0.0500,0
```

(a)

10	←	"neptune" was removed from the pattern and correlated to an attack type category number (that can be any number 1 to 23).

```
0,0.6667,0.7143,0.5455,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0.2407,0.0117,1,1,0,0,0.0500,0.0700,0,1,0.1020,
0.1000,0.0500,0,0,1,1,0,0
```

(b)

Figure 3. The two example packets in Figure 2 after processing for neural network displaying (a) a normal packet, and (b) an attack.

MATLAB Software Results

Prior to testing the chosen Multilayer Perceptron (MLP) algorithms for efficiency on low power systems, the MLP algorithm with two hidden layers was analyzed using MATLAB. Figure 4 shows the error minimization over several iterations of training. The y-axis is root mean squared error and the x-axis shows iterations of training. As shown in the figure, as training progresses, error decreases.

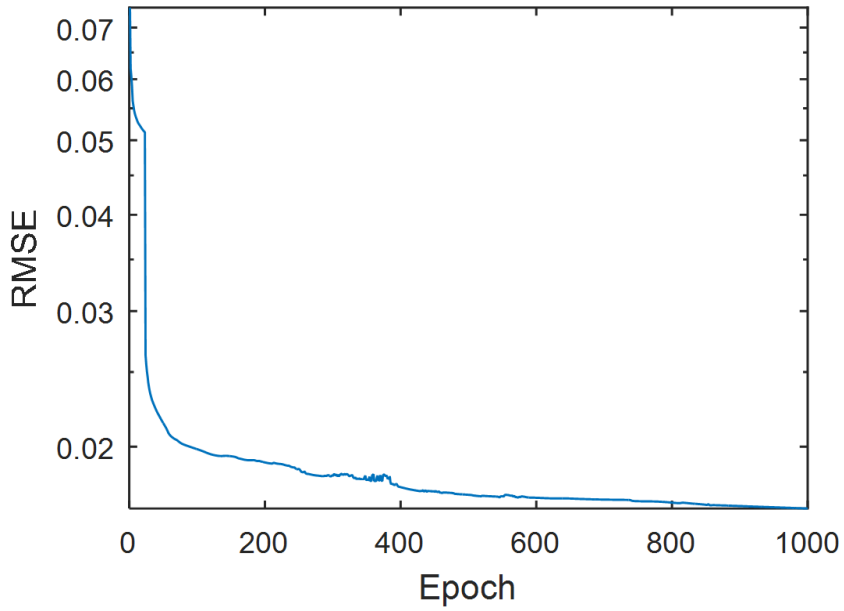


Figure 4. Root mean squared error minimization curve for training of multilayer perceptron algorithm with two hidden layers using MATLAB.

Following training, testing was completed. Table III shows the accuracy for various training data sizes and iterations of training. This chart shows that as the training data set size increases and as the number of iterations increases, accuracy generally increases. Generally, this increases in accuracy comes with an increase in time as well.

Table III. Accuracies during testing of multilayer perceptron algorithm with two hidden layers using MATLAB.

Iterations	Training Data Set Size					
	1000	**5000**	**10000**	**50000**	**100000**	**145586**
2	60.33	96.86	97.42	98.18	98.64	98.85
5	60.33	97.32	97.6	98.63	98.78	99.18
10	91.74	97.59	98.33	99.01	99.58	99.7
20	97.22	98.42	98.67	99	99.6	99.67
30	97.39	98.53	98.82	99.01	99.68	99.73
50	97.4	98.49	98.36	99.66	99.71	99.76
100	97.44	98.85	98.95	99.71	99.78	99.78

TensorFlow Software Results

Training

Table IV displays the two different perceptron topologies that were examined in this work. In each case, the network was trained for 100 epochs and the root mean squared error minimization curves are displayed in Figure 5.

Table IV. Different Multilayer Perceptron Topologies Used in this Work.

Network Type	Network Layout
MLP with 2 Hidden Layers	40→14→9→2
MLP with 1 Hidden Layer	40→14→2

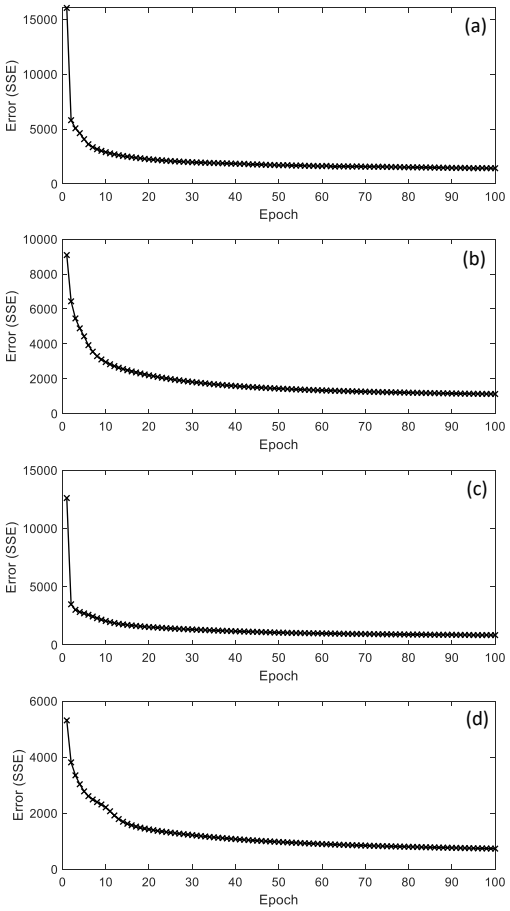


Figure. 5. Plots displaying training error over the 100 training iterations for a MLP with (a) two hidden layers using 100% of the data for training, (b) a single hidden layer using 100% of the data for training, (c) two hidden layers and 50% of the data for training, and (d) one hidden layer and 50% of the data for training.

The input layer had 40 entries in each case, because this is the number values that each data example contained after pre-processing the NSL-KDD dataset. Each of the MLP networks in Table IV were trained using two different data arrangements.

First, to directly study the networks' ability to train, all 125,973 packets in the dataset were used for training (see Figures 5 (a) and (b)). This same dataset is then used for testing. In the second data arrangement 50% of the packets were used for training and the other 50% were used for testing (see Figures 5(c) and (d)). This will test the networks ability to predict, as they will be tested on data that was not used during training. In each case in Figure 5, training correctly shows error minimization as the number of training epochs increases.

Testing and Evaluation

To judge each network layout in terms of how it can detect cyberattacks once training is complete, this section discusses testing accuracy. Tables V and VI show the testing results, and in each case these results represent an average of five identical training and testing executions. This provides a fairer measure as it accounts for the variation in the system, as weights will optimize differently with each run. Furthermore, in the case where 50% of the data was used for training, this 50% is randomly selected, and the testing and training datasets will differ for each run.

For the case where 100% of the data was used for training, that same data set was applied to each of the network configurations for testing and these results are displayed in Table V. When comparing the two different MLP layouts, the total accuracy is nearly identical. When looking at the classification breakdown in Table V, it appears that the system with one hidden layer is better at classifying attacks, and the system with two hidden layers has fewer false positives.

When comparing the data in Table V to that in Table VI, accuracy falls slightly when prediction is required to determine packet type, and this is to be expected. Again, the difference in accuracy between the MLP with one hidden layer and the MLP with two hidden layers is very small.

Table V. Classification Accuracy when Using 100% of the Data for Training.

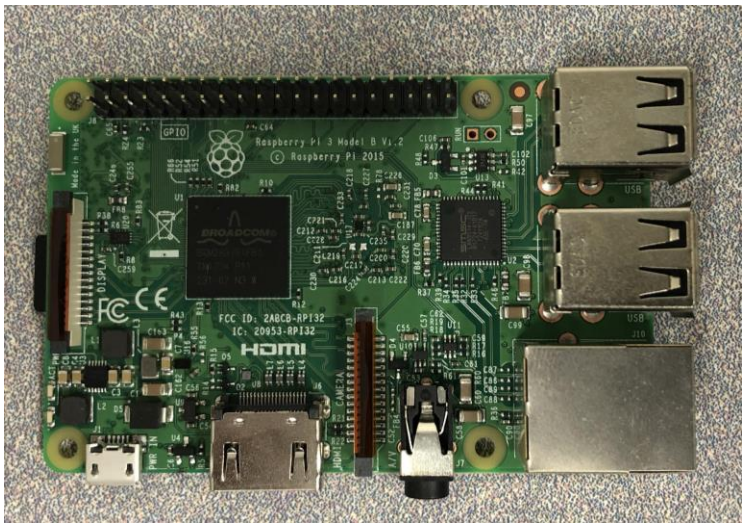
ANN Type	Accuracy	Hits	Correct Rejections	Misses	False Positives
Single Hidden Layer	99.458%	58226.6	67064.4	403.4	278.6
Two Hidden Layers	99.441%	58119.6	67111.8	510.4	231.2

Table VI. Classification Accuracy when Using 50% of the Data for Training.

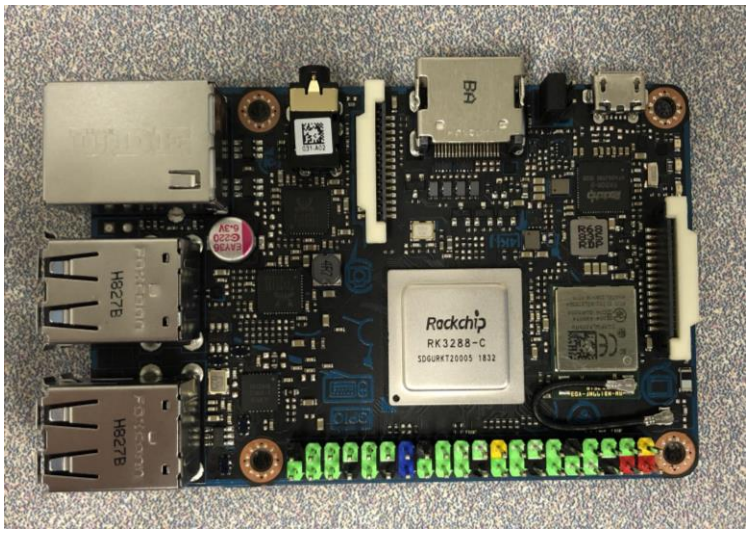
ANN Type	Accuracy	Hits	Correct Rejections	Misses	False Positives
Single Hidden Layer	99.190%	28828.6	33647.4	261.4	248.6
Two Hidden Layers	99.195%	28838.8	33640.2	251.2	255.8

Low Power Hardware Results

After the accuracy was analyzed for these networks, they were ported to two different minicomputer systems, the Raspberry PI 3 [13], and the Asus Tinkerboard [14]. Photos of each of these systems are displayed in Figure 6. The following subsections describe power, energy, and time analysis of these systems when executing the presented MLP networks.



(a)



(b)

Figure 6. Photographs of (a) the Raspberry PI 3 and (b) the Asus Tinkerboard.

Power Analysis

For each of the test cases described in Tables V and VI, the MLP networks were executed in Python scripts on each of these systems while a ‘watts up? PRO’ power analyzer logged power consumption at regular intervals during runtime. Figure 7 shows power consumption for each minicomputer when using the MLP with two hidden layers to train the entire dataset for a 20 epoch interval.

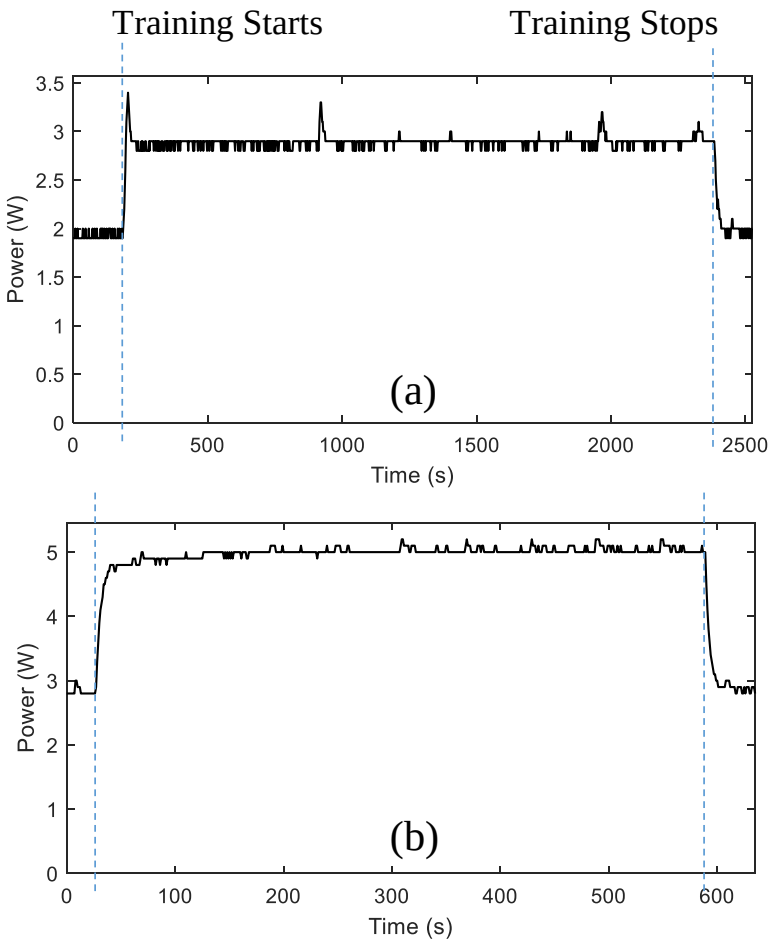


Figure 7. Power consumption during training when executing the MLP with 2 hidden layers on (a) the Raspberry PI and (b) the Tinkerboard (20 epoch training interval).

Likewise, Figure 7 shows the power consumption of each system when in network packet evaluation mode. To generate a result with a higher resolution time interval, the system was tested using the entire dataset 100 times.

Similar results were obtained for each of the MLP cases and were used to generate the analysis presented in the following tables. As in Table VII, we can subtract the idle power present in each system to determine the dynamic power requirement when executing these MLP scripts. Static Power was determined measuring each system's idle power for 300 seconds. The values and trends generated when testing these networks (as opposed to training them) are very similar so that data is not displayed. The differences between these systems in terms of energy is the more provocative result. Thus, the energy analysis of these systems is presented in Tables VIII through X.

Table VII. Power Consumption for Each of the MLP Systems During Training.

Power Measurement	**Raspberry PI**	**Tinkerboard**
Idle Power	1.7429 W	2.7337 W
Dynamic Power One Hidden Layer	0.9571 W	2.1292 W
Total Power One Hidden Layer	2.7004 W	4.8629 W
Dynamic Power Two Hidden Layers	1.1428 W	2.2366 W
Total Power Two Hidden Layers	2.8857 W	4.9703 W

Energy Analysis

Using the power consumption and runtime data collected during execution, energy consumption can also be determined for each of the MLP cases. Table VIII shows energy consumption for an entire training epoch for each case, where HL denotes the number of hidden layers in the network. Note that even though the Tinkerboard requires slightly higher power, its energy consumption is lower due to its higher execution speed. Table IX and X consider the per packet energy for training and testing respectively. The energy per packet is essentially the same for each MLP architecture whether 50% or 100% of the data is used for training because the networks are identical in each case. Due to the complexity of the training algorithm, an energy increase of nearly two orders of magnitude is present for a single packet when compared to testing.

Table VIII. Energy Per Training Epoch for Each of the Computing Systems Executing Each of the Four MLP Cases.

	Computing System	
Network	**Raspberry PI**	**Tinkerboard**
2HL 100%	316.759 J	278.983 J
2HL 50%	143.200 J	137.325 J
1HL 100%	191.099 J	191.319 J
1HL 50%	100.443 J	96.655 J

Table IX. Energy Per Packet During Training for Each of the Computing Systems Executing Each of the Four MLP Cases.

	Computing System	
Network	**Raspberry PI**	**Tinkerboard**
2HL 100%	2.514 mJ	2.215 mJ
2HL 50%	2.274 mJ	2.180 mJ
1HL 100%	1.517 mJ	1.519 mJ
1HL 50%	1.595 mJ	1.535 mJ

Table X. Energy Per Packet During Testing for Each of the Computing Systems Executing Each of the Four MLP Cases.

	Computing System	
Network	**Raspberry PI**	**Tinkerboard**
2HL 100%	37.494 µJ	29.738 µJ
2HL 50%	36.054 µJ	27.785 µJ
1HL 100%	28.402 µJ	22.990 µJ
1HL 50%	27.289 µJ	19.235 µJ

Time Analysis

Lastly, computation time per packet was examined for each of the MLP cases. These values correlate closely to the energy numbers for each of the minicomputer systems. However, timing data was also collected for a notebook computer running 64-bit Windows 10 Pro with an Intel Core i7-6700HQ CPU @ 2.6 GHz with 16 GB RAM to provide a baseline time value. In the future, we will also determine a fair way to measure energy consumption for traditional full sized computing systems.

When comparing Tables XI and XII, a consistent speedup of two orders of magnitude is observed when testing as opposed to training. This correlates to the energy values presented as well. To determine the speedup gained from a higher power system, a summary is presented in Table XIII. Each of the minicomputers are significantly slower than the Core i7 system. Thus, a user's hardware selection may be based on expected data throughput in addition to power requirements. For example, one of the low power systems could be more appropriate for a communication network that is used less frequently. The significant time and energy increases when moving from an MLP with a single hidden layer to one with two hidden layers should also be considered. These two systems did not provide a statistically significant difference in accuracy, but a 30% increase in cost (considering time and energy during testing) is required to add a second hidden layer to these networks.

Table XI. Execution Time Per Packet During Training for Each of the Computing Systems Executing Each of the Four MLP Cases.

	Computing System		
Network	**Raspberry PI**	**Tinkerboard**	**Core i7**
2HL 100%	871.365 µs	445.571 µs	69.389 µs
2HL 50%	854.989 µs	450.356 µs	69.143 µs
1HL 100%	561.846 µs	312.310 µs	49.678 µs
1HL 50%	600.609 µs	314.719 µs	50.689 µs

Table XII. Execution Time Per Packet During Testing for Each of the Computing Systems Executing Each of the Four MLP Cases.

	Computing System		
Network	**Raspberry PI**	**Tinkerboard**	**Core i7**
2HL 100%	14.564 µs	6.063 µs	0.817 µs
2HL 50%	14.422 µs	6.157 µs	0.690 µs
1HL 100%	11.067 µs	4.459 µs	0.798 µs
1HL 50%	11.004 µs	4.413 µs	0.600 µs

Table XIII. Time Factors Showing How Much Speedup is Obtained for Each System When in a Testing Mode.

	Computing System		
Network	**Raspberry PI**	**Tinkerboard**	**Core i7**
2HL	17.83×	7.42×	1×
1HL	13.87×	5.59×	1×

Conclusion

This work examines operation of a perceptron based intrusion detection system implemented on low power hardware. A power, energy, timing, and accuracy design space analysis was performed to quantify the benefits of portable low power intrusion detection. Greater than 99% accuracy is achieved using the proposed multilayer perceptron algorithm. Using low power hardware, a scan rate of greater than 226,000 packets per second can be achieved while consuming less than 5 Watts of power.

We have several ideas for future work including a more complete power analysis that includes optimized desktop hardware. We would like to further examine the relationship between network complexity and accuracy for network intrusion detection in order to develop a relationship between network power consumption and accuracy. We would also like to implement alternative neural network algorithms that may be more suited to recognizing zero day attacks that utilize in-situ training.

References

[1] I. Ahmad, A. B. Abdullah, and A. S. Algamdi, "Application of Artificial Neural Network in Detection of Probing Attacks" IEEE Symposium on Industrial Electronics and Applications (ISIEA 2009), October 4-6, 2009, Kuala Lumpur, Malaysia.

[2] D. Wu, L. Pigou, P.-J. Kindermans, N. D.-H. Le, L. Shao, J. Dambre, and J.-M. Odobez, " Deep Dynamic Neural Networks for Multimodal Gesture Segmentation and Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 8, pp. 1583-1597, Aug. 2016.

[3] Z. Si, H. Yu, and Z. Ma, "Learning Deep Features for DNA Methylation Data Analysis," IEEE Access, vol. 4, pp. 2732-2737, June, 2016.

[4] P. K. Prajapati and M. Dixit, "Un-Supervised MRI Segmentation using Self Organised Maps," International Conference on Computational Intelligence and Communication Networks, pp. 471-474, Dec. 2016.

[5] C. Yakopcic, M. Z. Alom, and T. M. Taha, "Memristor Crossbar Deep Network Implementation Based on a Convolutional Neural Network," IEEE IJCNN, 2016.

[6] S. Wang, W. Wang, C. Yakopcic, E. Shin, G. Subramanyam and T. M. Taha, "Reconfigurable Neuromorphic Crossbars Based on Titanium Oxide Memristors," Electronics Letters (Accepted).

[7] C. Yakopcic, R. Hasan, T. M. Taha, M. R. McLean, and D. Palmer, "Memristor-based neuron circuit and method for applying learning algorithm in SPICE," Electronics Letters, vol. 50, no. 7, pp. 492-494, 2014.

[8] C. Yakopcic, R. Hasan, T. M. Taha, and D. Palmer, "SPICE Analysis of Dense Memristor Crossbars for Low Power Neuromorphic Processor Designs" IEEE National Aerospace and Electronics Conference, June, 2015.

[9] C. Yakopcic, R. Hasan, and T. M. Taha, "Memristor Based Neuromorphic Circuit for Ex-Situ Training of Multi-Layer Neural Network Algorithms," IEEE IJCNN, 2015.

[10] T. M. Taha, R. Hasan, and C. Yakopcic, "Memristor Crossbar Based Multicore Neuromorphic Processors," IEEE International SOC Conference, 2014.

[11] https://www.unb.ca/cic/datasets/nsl.html, Canadian Institute for Cybersecurtiy, University of New Brunswick.

[12] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, University of California, Irvine, Irvine, CA, Oct. 1999.

[13] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[14] https://www.asus.com/us/Single-Board-Computer/Tinker-Board/

[15] R. Vinayakumar, K. P. Soman, P. Poornachandran, "Applying Convolutional Neural Network for Network Intrusion Detection," IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1222-1228, Sept. 2017, Udupi, India.

[16] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," IEEE Access, vol. 7, pp. 41525-41550, April 2019.

[17] S. Siddiqui, M. S. Khan, K. Ferens, "Multiscale Hebbian Neural Network for Cyber Threat Detection," International Joint Conference on Neural Networks (IJCNN), pp. 1427-1434, May, 2017, Anchorage, AK, USA.

[18] F. Palenzuela, M. Shaffer, M. Ennis, J. Gorski, D. McGrew, D. Yowler, D. White, L. Holbrook, C. Yakopcic, and T. M. Taha, "Multilayer Perceptron Algorithms for Cyberattack Detection," IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), pp. 248-252, OH, July 2016.

[19] M. Z. Alom and T. M. Taha, "Network Intrusion Detection for Cyber Security on Neuromorphic Computing System," International Joint Conference on Neural Networks (IJCNN), pp. 3830-3837, May, 2017, Anchorage, AK, USA.

[20] M. S. Alam, B. R. Fernando, Y. Jaoudi, C. Yakopcic, R. Hasan, T. M. Taha, and G. Subramanyam "Memristor Based Autoencoder for Unsupervised Real-Time Network Intrusion and Anomaly Detection," International Conference on Neuromorphic Systems (ICONS), 2019 (Accepted).

[21] J. Sklansky and L. Michelotti, "Locally Trained Piecewise Linear Classifiers," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 2, no. 2 pp. 101-111, March, 1980.

[22] Parallel back-propagation neural network training technique using CUDA on multiple GPUs," IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), pp. 1-3, Aug. 2015.