Virus Recognition Based on Combination of Hashing and Neural Networks
Mohamed H. Almeer
Pages 237 – 242
http://dx.doi.org/10.21700/ijcis.2016.128

Virus Recognition Based on Combination of Hashing and Neural Networks

Mohamed H. Almeer
Computer Science & Engineering Department,
Qatar University
Doha, QATAR
almeer@qu.edu.qa, almeerqatar@gmail.com

Abstract: *In this paper, we propose an intelligent first-warning system for virus code detection based on Artificial Neural Networks (ANNs). The proposed system operates in accordance with the basic principles of ANNs to conduct pattern matching of 32-bit hash signatures and detect virus signatures by means of the hashing applied to the byte content of executable code. The proposed system can accurately detect virus code in accordance with information it has learned, and gives false positive ratios within acceptable ranges. The results of experiments conducted show that the combination of 32-bit hashing and neural networks results in a low false positive rate. This paper also discusses the key ideas and approaches, along with the necessary adaptations and adjustments undertaken in the neural network model underlying the proposed early warning virus detection system..*

Keywords: *Hashing, Hash code, BKDR hash function, ANN, Neural Networks, Virus detection.*

Received: July 30, 2016 | ***Revised:*** August 10, 2016 | ***Accepted:*** August 25, 2016

1. Introduction

Most commercial antivirus software products currently available depend on signature-based virus detection and heuristic classifier models with the ability to detect new viruses. However, the 'classic' signature-based detection algorithm simply uses the byte signatures of known viruses saved in memory to generate detection models. In general, detection methods based on byte signatures use a large collection of regular expressions or simple signature-string-matching engines to scan files. In addition to antivirus software products, packet processing applications that inspect packets at a level beyond protocol headers need to analyse the contents for some known signatures. For instance, network security applications must neglect packets containing certain harmful internet viruses and worms carried in packet payloads.

In general, payload-scanning applications and antivirus software have a common requirement for string matching. In addition, the location (or offset) of such suspicious strings need to be sensed in the packet payload because their length is anonymous; therefore, such applications must have the capability to detect strings of various lengths, starting at arbitrary locations in the packet payload.

We propose a hardware-based technique in which bloom filters are integrated with an Artificial Neural Network (ANN) to detect strings in streaming data, which is both useful for antivirus applications and sufficient for network packet content inspection (Figure 1).

Figure 1. Model of the proposed HASH-ANN system

We chose a bloom filter to identify signatures because of its data structure, in which the computation time involved in performing the query is completely independent of the number of strings in the database, given that the memory used by the data structure scales linearly with the number of strings stored in it. In addition, the amount of storage

required by the bloom filter for each string is independent of its length.

Our implementation groups signatures in bytes according to a sliding window, applies a hash function to them, and stores each string's hash in four bloom filters. Each bloom filter scans the streaming data and checks the strings. Whenever all bloom filters simultaneously detect a suspicious string, an analyser probes the string to decide whether it indeed belongs to the given set of strings or it is a false positive.

Tan [1] proposed a traditional multilayered feed forward network for automatic intruder detection in UNIX systems. James and James [2] utilised an ANN for misuse detection and analysed the applicability of neural networks in the identification of instances of misuse against a network. They used the ANN to identify and classify network activity based on limited, incomplete, and nonlinear data sources. Arnold and Tesauro [3] automatically constructed multiple neural network classifiers which can detect unknown Win32 viruses. They constructed heuristic classifiers with the ability to distinguish between uninfected and infected members of some class of program objects. Salameh [5] used packet filtering software and nine identifiers from a network packet frame to classify various malicious patterns that signify the presence of malicious codes. Dahl et al. [6] used approximately 2.6 million virus samples to train a very large neural network via feature selection, which reduces the number of general features to manageable numbers.

Research has also been conducted on the implementation of hash functions using neural networks. For example, Lian et al. [9] presented and analysed a secure hash function based on a neural network. The hash function adopted the neural network's one-way property and was used for security applications. Although the concept of bloom filter, which in essence uses the hashing for large information coding, has been generally addressed, there is no indication of it being used with neural network systems to implement hashing primarily for malicious code identification. In this paper, we elaborate on the missing concepts, and show how 50 viruses are coded using 32-bit hashing implemented through neural networks. Experimental results are also presented.

2. Bloom Filters

A bloom filter is a space-efficient data structure that is used to test whether an element is a member of a set. With bloom filters, false positive matches are possible, but false negatives are not. Thus, a bloom filter has the property that a query either returns "Yes, possibly included" or "definitely no, not included". Elements can easily be inserted into the set, but not possibly removed. The more elements inserted into the set, the larger the probability of false positives becomes.

To query the filter using a test element to determine whether it is in the set or not, we can feed it to each of k hash functions (but we used only one in our study) to look up k arrays. If any of the resulting bits from the arrays is zero, the element is definitely not in the set. However, if all are, then, either the element is in the set (which is one of the expectations), or the bits have a mistaken incident, resulting in a false positive. Unfortunately, there is no way to distinguish between the two cases.

The requirement that k different independent hash functions be designed can be difficult for large k. For a good hash function with a wide output, there should be little if any correlation between different bit-fields of such a hash, so this type of hash can be used to generate multiple 'different' hash functions by slicing its output into multiple bit-fields.

3. Objective of the Research

The objective underlying Hash-ANN is as follows. Currently, memory access requires a few cycles in order to acquire the required data, while going through a neural network structure can cost less if carried out using the current fast FPGA technology available nowadays, and hence totally avoiding the use of memory units.

Bloom filters also have an unusual property whereby the time needed either to add items or to check whether an item is in the set is a fixed constant, O(k) (The order is linearly related to the number of hashes k), which is completely independent of the number of items already in the set. No other constant-space set data structure has a similar property.

This property also makes it a good support system for neural network cases in which the processing time needs to be as small as possible for fast execution. Secondly, in the conventional bloom filters, the scanning algorithm uses k independent hash functions working on strings of bytes and constructs a bloom filter from the virus signature set. However, in our algorithm, the k hashes are replaced with the split of a single 32-bit hash into 4 Bytes that each probes through its own bloom filter (i.e., the hash results are used as indices to set bits in the bloom filter bit array to one).

The third property that distinguishes the current algorithm from conventional algorithms is its parallel processing nature which suitable to be implemented using FPGA technology. The scanning algorithm, where the software implementation is considered, applies the hash functions one at a time. After one function, the algorithm checks the bit in the bloom filter. If the bit is one, it goes on to the next hash function; otherwise, it immediately slides the window down one byte and starts again processing the next m-bytes block. This process is completely avoided in

our approach because the ANN circuit actually computes the four-hash using all the weights of the neural network simultaneously, and at the end sees if the four results are fully 1.

4. Selecting Bloom Filter Function and Width

We then proceed to add the Brian Kernighan and Dennis Ritchie (BKDR) hash function [11]. The function is a simple hash function that uses a strange set of possible seeds which all constitute a pattern of 31, 131, 1313, 13131, 131313, etc., which is very similar to the DJB hash function. A C Language implementation of the algorithm, in which the variable type has been chosen to be a 32-bit 'unsigned integer', is outlined below. Here we consider the inputs as the series of bytes needed to compute the hash for (*input[i]-variabble)* and the output is the (*hash*-variable) as a 32-bit integer.

```
unsigned int seed = 131;  // 31 131 1313 13131 131313 etc..
unsigned int hash = 0;
for (int i = 0; i < size; i++)
  {
        hash = (hash * seed) + static_cast<unsigned
        short>(input[i]);
  }
return hash;
```

5. Selecting Bloom Filter Size

Traditional bloom filter implementations select a filter size such that it has approximately one-half of the filter when it is populated. Because four variations of the signature are inserted into the filter, for a virus signature database comprising N signatures, and also because four variations of hash functions are set, a bloom filter of length 32×N bits, or 4×N bytes, is considered sufficiently large. Based on this derivation, a 256X1 filter should be used for a maximum of 64 signatures. In our study, we chose 50 signatures to populate the bloom filter table.

6. NN-Hash function

Our NN-Hash function is considered only during the first-pass scan on data to determine if the data needs to go through an exact-match algorithm or not, without actually carrying out the latest process. The algorithm moves a sliding window n bytes down the input stream. For each byte under the window, (32-bit hash value = 4 Bytes) a hash function is applied resulting in their hash values. The four-byte hash results are then used to probe a bit array of 256 elements each, which is a bloom filter pre-constructed from the virus signatures.
At scan time and for ordinary hash tables, for each of n bytes under the sliding window, the same hash functions are applied to them, and the results are probed in the bloom filter. If all bits are one, the exact-match algorithm (for example [14]) is invoked to determine if there is an actual match with a virus signature. Meanwhile, in the NN-Hash approach, the last phase is completely substituted by going through a neural network in one pass, and the result should indicate the presence of a virus if it has been recorded in the training set.

7. Artificial Neural Networks (ANNs)

An ANN comprises a number of interconnected processing elements and maps a set of inputs to a set of desired outputs. The characteristics of the elements and the weights associated with the interconnections among them determine the result of the transformation. The nodes of the network are able to easily adapt to the desired outputs by changing the connections that link them. An ANN conducts an analysis of the information it is given and outputs the results as a probability estimate of the data matching the required pattern which it has been trained to recognise.
This characteristic makes ANNs among the most desired methods in pattern recognition and signature matching. However, the accuracy of the match and the decisions relies completely on the experience of the system (the memorization process) cultivated during the training phase using examples of the stated problem. Fig. 2 depicts a typical multilayer feed forward ANN.

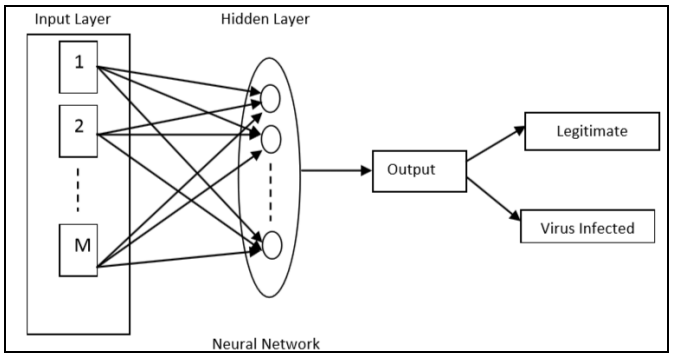

Figure 2. Typical multilayer ANN circuit

8. Data preparation and training

***ClamAV Background*:** ClamAV [13] is one of the most widely used open source virus scanners. We used the signatures from ClamAV in our study. ClamAV o□ers client-side protection for personal computers, as well as mail and file servers for large organisations. The ClamAV virus database is updated at least once every four hours and, as of 25 December 2014, contained over 3,700,000 virus signatures with the daily update virus DB number at 19,837s. In addition, it has a core scanner library and command-line utilities.

The ClamAV database contains signatures for non-polymorphic viruses in simple string format, and for polymorphic viruses in regular expression format.

The current version of ClamAV uses a summarised or simplified version of the Boyer-Moore algorithm [16] to detect non-polymorphic viruses via simple fixed-string signatures. It uses a variant of the classical Aho-Corasick and Wu-Manber algorithms for polymorphic viruses. [14], [15].

Because of the simple format in which it stores its virus signatures, and its user-friendly conversion utility, we were able to extract from its database of patterns the needed N-gram signatures that we utilised to train the ANN in our proposed system.

Four C language programs were developed: to assist in the extraction of the required N-gram signatures corpus of the virus body, to pre-process the data to make them available for training in the MATLAB neural network toolbox, and to prepare the patterns from collections of executable files.

1. The first program processes the main ClamAV signature database file and converts the content from binary format to hex format in ASCII. The 'main.cvd' database file in which ClamAV stores all its signatures holds the static signature of 3.7 million viruses.
2. The second program is responsible for extracting more details about the virus corpus and deleting or bypassing head identification data, such as the virus name.
3. The third program is responsible for choosing the specific number of virus N-grams, specified by the user, in any sequence, then applying the hash function to its contents, which is further processed by the neural network toolbox in MATLAB.
4. The fourth program reads a collection of Windows executable files with sliding window effects, then applies the hash function. The resulting four bytes are saved in a generated file. This preparation is essential in order for MATLAB to apply those collections of patterns to the neural network for validation. Nevertheless, this maximizes the size of the generated files fourfold; for instance, a file such as 'adamsync.exe', with a size of 164 KB, will generate text files 656 KB in size, regardless of the window size (N-gram).

Neural Networks as a Pattern Recognition Model: Virus code detection can be viewed as a binary classification problem. Thus, we can use a multilayer ANN as a pattern matcher to carry out the detection process. The details of the proposed detection procedure can be summarised in the following steps:

- Dump hexadecimal byte sequence from viruses taken from the ClamAV main virus database, 'main.cvd', and benign install '.exe' executable files that exist in any PC.
- Slice each hex sequence into n grams—the size of the sliding window—and apply a hash function to them. Then, save the resulting four-byte hash value in a file to be presented to the trainer.
- Conduct training and validation of the model.

Tables 1 and 2 present the details of the contents of one of the viruses used in ClamAV and its format as seen in the ClamAV main database file, and the names of the 50 viruses used in our study, respectively.

The most frequently used ANN is the multilayer feed forward ANN. We used it in our study because it is considered the workhorse of neural networks in general. It can be used for both function fitting and pattern recognition applications (as in our case).

Table 1. Virus Signature format in ClamAV and its reduced content used in the study for N=16.

Virus Name	Trojan.Proxy.Ranky-45
Detailed pattern in 'main.cvd'	Trojan.Proxy.Ranky-45:1:*:c6d7435624d832415cf0b5eeaf5b302f5765319e08957b8404f92f956381e4f33f56e26e9ceff8f2a8eec259dd89af336911a7564a2b0802779a3a2b5beb9bc576e3610e405823faff872cab2268b5b9e8c4401a8747941015e36680762559c8bc948b523d27ce29a700e5dbe2543f7904174250d54b3cb1633a976458d0d5070f0ce5837466603aa4eb575e7d622bf8465271ef14
Extracted and summarised pattern used in the study	c6d7435624d832415cf0b5eeaf5b302f5765319e08957b8404f92f956381e4f33f56e26e9ceff8f2a8eec259dd89af336911a7564a2b0802779a3a2b5beb9bc576e3610e405823faff872cab2268b5b9e8c4401a8747941015e36680762559c8bc948b523d27ce29a700e5dbe2543f7904174250d54b3cb1633a976458d0d5070f0ce5837466603aa4eb575e7d622bf8465271ef14
N-grams = 16 Bytes	c6d7435624d832415cf0b5eeaf5b302f

Table 2. First 50 viruses found in ClamAV and used in the study.

N	**Virus Name**	**N**	**Virus Name**
1	Exploit.HTML.ObjectTy	26	HTML.Phishing.Bank-27
2	Exploit.HTML.DragDro	27	HTML.Phishing.Bank-23
3	HTML.Phishing.Bank-4	28	HTML.Phishing.Bank-25
4	W32.MyLife.E	29	HTML.Phishing.Bank-26
5	Trojan.Downloader.Smal	30	Trojan.Banker-7
6	HTML.Phishing.Auction	31	HTML.Phishing.Bank-29
7	HTML.Phishing.Auction	32	HTML.Mydoom.email-gen-1
8	HTML.Phishing.Bank-5	33	Trojan.Spy.Flux.A-srv
9	HTML.Phishing.Bank-6	34	Trojan.Downloader.Small-167
10	W32.Wabrex.A	35	HTML.Mydoom.email-gen-2
11	HTML.Phishing.Bank-8	36	HTML.Mydoom.email-gen-3
12	HTML.Phishing.Bank-9	37	HTML.Mydoom.AD-exploit
13	HTML.Phishing.Bank-	38	HTML.Phishing.Bank-32
14	HTML.Phishing.Bank-	39	HTML.Phishing.Bank-31
15	Trojan.Downloader.VBS	40	HTML.Phishing.Bank-34
16	Trojan.Flashkiller.A	41	HTML.Phishing.Bank-35
17	HTML.Phishing.Bank-	42	Exploit.HTML.IFrameBOF-1
18	HTML.Phishing.Bank-	43	HTML.Phishing.Auction-3
19	HTML.Phishing.Bank-	44	Trojan.Downloader.Psyme-2
20	Trojan.Downloader.JS.P	45	Trojan.Downloader.Monurl-1
21	Trojan.HacDef-1	46	HTML.Phishing.Bank-38
22	Trojan.Regger-1	47	HTML.Phishing.Bank-37
23	HTML.Phishing.Bank-	48	HTML.Phishing.Bank-36
24	HTML.Phishing.Bank-	49	HTML.Phishing.Auction-4
25	HTML.Phishing.Bank-	50	Trojan.Downloader.Small-169

The ANN used in our study comprised an input layer and an output layer. To standardise the comparison of

different approaches used to test the recognition of virus patterns, we froze some properties of the ANN: the number of layers (set at two, one input and one output), the number of neurons in the input layer is the N-gram used for the size of the sliding window, the number of neurons in the input layer was set at 500, and the number of neurons in the output layer was set to one because it signals true or false for the virus presence or absence. Table 3 presents the details of this arrangement. The results of prior experiments conducted indicate that the two-layer network is sufficient for fast training and successful recognition.

Table 3. Neural Networks characteristics.

Property	**Value (Range)**	
Number of layers	3 (Input. hidden and Output)	
Neurons and activation function (per layer)	**Layer 1,2**	(1, 500) Sigmoidal
	Layer3	(1) Linear
Window size (N-grams) Input layer size	3-4-6-8-10-12-14-16	
Training algorithm	(RP) TRAINRP Resilient Backpropagation.	
Training error goal	1.0E-4	
Max epochs allowed (given)	100,000 Iterations	
Computer (OS) used	Lenovo Intel® Core i3-3217U CPU, 1.8 GHz, 4 GB RAM, 64-bit OS, X64-based processor (Windows-8.1 Pro)	

Sigmoidal activation functions are used in both the input layer and a linear activation function in the output layer. Fig. 2 depicts the ANN used in our study.

The TRAINSCQ and TRAINRP training algorithms are recommended for training large networks, and pattern recognition networks in particular. The memory requirements of these algorithms are relatively small, and yet they are much faster than standard gradient descent algorithms. However, TRAINRP is more suitable for pattern recognition, and is good enough for large networks with many neurons, and large datasets because it is the fastest known algorithm. Our study utilises more than 500 neurons and a large number of training sets, which justifies the use of 'TRAINRP,' or the 'back propagation algorithm.

Table 4. Neural Networks recognition results for training error goal = 1.0E-4 and network 500-1 neurons.

N	**False Positive (%) Hash-ANN**	**False Positive (%) N-Grams ANN**
3	0.553	N.A.
4	0.1282	2.88
5	0.0504	N.A.
6	0.0800	0.884400
8	0.0698	0.914700
10	0.0740	0.212300
12	0.0771	0.149200
14	0.0996	0.057500
16	0.0895	0.031100

9. Experiments

We utilised the main virus signature database of the ClamAV program, which we downloaded from its website [13]. The collection comprises more than 100,000 static virus signatures and approximately 750,000 MD5 hashed patterns. Only 50 virus codes were extracted.

We collected more than 125 win32 executable files from a fresh installation of Windows 8, in addition to other applications installed, on an ordinary Windows-based PC. The total size of the files was 54 MB. We used the files collected to test and validate the ANN. In order to efficiently use the validation method, we focused our attention on the false positive rate as a performance indicator.

The output of the HASH-ANN is never wrong unless there is a collision between two different input patterns giving the same hash value—a possibility that is extremely low. More specifically, the training of each bloom filter with 256 entries of 0's and 1's was completed when the network converged to the required error rate (0.0001). Thus, when presented with any of the 256 values for a byte, it would immediately give the correct answer in accordance with its training. Therefore, the network will trigger a virus recognition only when the output is greater than 0.5, leaving no place for any error.

We tested the validity of the network by conducting an experiment in which the number of N-grams was increased from three to 16, while the number of virus codes to be recognised was set at 50 viruses. In previous work, we were able to recognise 50 viruses using neural networks and mimicking N-grams [17]. In the previous work, an ANN was used to recognise complete N-patterns taken from the body of some known viruses. Fig. 3 shows the results obtained in that case. The horizontal axis indicates the number of N-grams used in both studies, while the vertical axis shows the false positive ratio (in percentage).

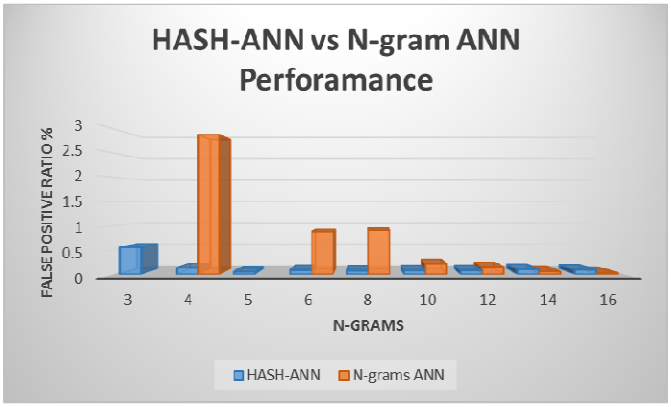

Figure 3. Graph showing the change in false positives for various N-values in the N-grams for HASH-ANN and N-gram ANN

The results (shown in Table 4) are very encouraging, because a False Positive (FP) ratio of 0.1282% for the

configurations of N=4 of a window size is shown to have been achieved; whereas in previous work [17], the lowest FP achieved was FP = 2.88%. Thus, our current result is 22 times better. Other N-values can deliver more evidence to improve the accuracy of this method compared to the normal N-gram identification. However, for large N-values (N > 14), the two approaches have virtually the same accuracy ratio.

10. Conclusion

In this paper, we presented encouraging preliminary results for the application of feed forward neural networks based on byte N-gram and hash functions in the detection of virus codes using virus patterns extracted from ClamAV, the most well-known free and open source antivirus program. The method achieves an FP rate of 0.0895% for N-gram > 16 on training data, and 0.553% for N-gram = 3, based on the training condition and error goal of 1.0E-4. In future work, we plan to conduct experiments on larger data collections while reducing the size of the network (fewer neurons). In addition, we will consider implementation of this method using FPGA or reconfigurable logic.

References

[1] K. Tan, "The application of neural networks to UNIX computer security," Proceedings of the IEEE International Conference on Neural Networks, vol. 1, 1995.

[2] C. James and M. James, "The application of artificial neural networks to misuse detection: Initial results," Proceedings RAID98, Louvain-la-Neuve, Belgium, pp. 14-16, 1998.

[3] W. Arnold and G. Tesauro, "Automatically generated Win32 heuristic virus detection," Virus Bulletin Conference, pp. 51-60, September 2000.

[4] G. J. Tesauro, O. J. Kephart, and B. G. Sorkin, "Neural networks for computer virus recognition," IEEE Expert Magazine, pp. 5-6, 1996.

[5] W.A. Salameh, "Detection of intrusion using neural networks: A customized study," Studies in Informatics and Control, vol. 13, no. 2 pp. 135-143, 2004.

[6] G.E. Dahl, et al. "Large-scale malware classification using random projections and neural networks," Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.

[7] S. Shah, et al. "Virus detection using artificial neural networks," International Journal of Computer Applications, vol. 84, 2013.

[8] V. Golovko and S. Bezobrazov, "Neural network artificial immune system for malicious code detection," ICNNAI'2010, pp. 147-153, June 2010.

[9] S. Lian, J. Sun, and Z. Wang, "One-way hash function based on neural network," arXiv preprint arXiv:0707.4032, 2007.

[10] O. Haddadi, Z. Abbasi, and H. TooToonchy, "The Hamming code performance analysis using RBF neural network," Proceedings of the World Congress on Engineering and Computer Science, vol. 2, 2014.

[11] B.W. Kernighan and D.M. Ritchie, The C Programming Language, Prentice Hall Professional Technical Reference, 1988, ISBN:0131103709.

[12] Neural Network Toolbox User Guide.

[13] ClamAV, www.clamav.org.

[14] A.V. Aho and M.J. Corasick, "Efficient string matching: An aid to bibliographic search," Communications of the ACM, vol. 18, no. 6, pp. 333-340, 1975.

[15] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," Technical Report TR-94-17. University of Arizona, 1994.

[16] R.S. Boyer and J.S. Moore, "A fast string searching algorithm," Communications of the ACM, vol. 20, no. 10, pp. 762-772, 1977.

[17] M.H. Almeer, "N-grams and neural networks in early virus warning," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol. 4, no. 4, pp. 1-5, April 2015.

Dr. Mohamed H. Almeer is an Assistant Professor in the Department of Computer Science and Engineering, College of Engineering, Qatar University. Qatar. He acquired his PhD in Computer Engineering in 2000. His main research focus is Microprocessors, Neural Networks, Digital Design based on PLD devices, and Co-Design