Hopfield Neural Networks as Pseudo Random Number Generators

Kayvan Tirdad
Computer Science Department
Ryerson University
Toronto, Canada
Kayvan.Tirdad@Ryerson.ca

Alireza Sadeghian
Computer Science Department
Ryerson University
Toronto, Canada
Asadeghi@Ryerson.ca

***Abstract*—Pseudo random number generators (PRNG) play a key role in various security and cryptographic applications where the performance of these applications is directly related to the quality of generated random numbers. The design of such random number generators is a challenging task. In this paper, we propose an application of Hopfield Neural Networks (HNN) as pseudo random number generator. This is done based on a unique property of HNN, i.e., its unpredictable behavior under certain conditions. We compare the main features of ideal random number generators with those of PRNG based on Hopfield Neural Networks. We use a battery of statistical tests developed by National Institute of Standards and Technology (NIST) to measure the performance, and to evaluate the quality of the proposed Hopfield random number generator.**

Keywords; Psudo Random Number Generatore, Hopfield Neural Networks,Security

I. INTRODUCTION

Random number generators are widely used in cryptosystems, where existence of random numbers is essential for example in generating of keys. The random number generators used in cryptographic applications typically produce a sequence of bits consisting of zeros and ones which can be combined into blocks or subsequence of random numbers in two basic classes called deterministic and nondeterministic. The deterministic random number generators produce a sequence of bits from an initial value, i.e., seed, with a specific algorithm. The nondeterministic generators produce unpredictable outputs that depend on some physical sources such as noise in different signals [1].

One of the major concerns in cryptographic applications is to develop a Pseudo Random Number Generator (PRNG) with acceptable quality in term of randomness. There are a large number of PRNG algorithms reported in the literature [1]. There also exist different procedures to measure the quality of PRNGs in terms of the randomness where a random process is defined as one whose consequences are unknown [2]. With the advent of more powerful computational devices in recent years, weaknesses in generated random sequences can be more easily exploited. Therefore, there is an increasing need for both stronger PRNGs and more accurate randomness tests [3], [4]. This paper deals with the design of pseudo random number generators that are based on Hopfield neural networks.

Recurrent Neural Networks have been used in different Cryptosystems, such as symmetric cipher [5]-[7]. Neural networks have number key features that make them suitable mechanism to work as random number generators. These properties include complexity, dynamic and parallel distributed structure, nonlinearity, and unpredictability. Different types of neural networks have already been used as random number generators in cryptosystems [8], [9]. In particular, application of Hopfield neural networks to pseudo random number generators have also been studied in [9], [10].

Hopfield Neural Networks (HNN) possesses function approximation and generalization capabilities that together with instability and non-convergence properties can be shown to be advantageous when dealing with the generation of random numbers.

A recurrent network with no self-feedback, Hopfiled neural network convergence is based on an energy function that is decreasing over time [11]. The underlying idea then is that if non-convergence can be guaranteed, HNNs can be used as random number generators. This is because the network output can be shown to be unpredictable before the network converges [9].

In what follows, the randomness concept and random number generators test will first be discussed in Section II. We will discuss the RNG tests and specially examine NIST RNG test suit. The importance of this test will be highlighted. In section III, Hopfield neural networks and their stability problem will be discussed. We will explain our HNN based non-converge approach and its application to PRNG. The results of our experiments showing the strength of the proposed PRNG is given in Section IV. In Section V, we will provide discussion about improvement of our proposed PRNG in terms of its randomness. The conclusion is provided in section VI.

II. TEST OF RANDOMNESS

A numeric sequence that contains no patterns and regularities is known as statistically random sequence. When a number is chosen arbitrary from some specific distribution it can be called a random number. Such numbers are almost expected to be independent with no correlations with successive numbers. Random numbers generated by computers are called pseudo random numbers. The term

978-1-4244-7858-3/10/$26.00 ©2010 IEEE

"random" is usually used when the output of a process is unpredictable. However, when randomness is used within the context of uniform distributions, it means that random sequences are generated in an almost predictable fashion using some mathematical formula and stochastic process, but the distribution area is large enough and normalized, and hence the prediction is almost impossible [12].

To check the result of the random number generators and evaluate their suitable for a particular cryptographic application, a number of tests exist to check the randomness and unpredictability of the results of software or hardware based random number generators [13]. Statistical tests are used widely [14], [15] and provide good measurements for randomness. For example, National Institute of Standards and Technology (NIST) provide statistical standards such as NIST Test Suite that measures the randomness degree of binary sequences. NIST Test Suite consists of a number of tests that together search for different types of non-randomness which might exist in the binary sequences [14].

Each test provides a randomness measure, i.e., *P-value*. There are two general parameters that we should set before using this test. First one is α that defines the limitation of acceptance of the result. For example, if we set α to 0.01, it means that number of sequence that we use is 100 and only if one or zero of them is rejected we consider that the result is random. The other parameter is the length of the each sequence, NIST suggests, that it must be at least 10^6.

A. Frequency Test

The purpose of the test is to focus on the proportion of zeroes and ones in whole sequence, whether the number of zeros and ones are approximately the same, as it would be expected for a random sequence. The outcome of the test is a parameter called *P-value* that represents the ratio of the abstract of the summation of every binary numbers of the sequence in the way that the zeros are converted to value of -1 to the square root of the number of the whole sequence. Thus, *P-value* indicates how random the sequence is [14].

B. Frequency Test within a block

This test relates to the proportion of ones in *M*-bit blocks, where *M* denotes the length of each block. This test determines whether the observed proportion of ones, within *M*-bit block, meets the expected proportion and the frequency of ones is approximately *M*/2 [14].

C. Runs Test

The test calculates the total number of the identical bits bounded with a bit of the opposite value before and after them, in an uninterrupted sequence. This series of identical bits is called Runs, and the purpose of this test is to determine whether this number of Runs with a various length is as expected for a random sequence, or how fast is the oscillation between these ones and zeros [14].

D. Test for the Longest Run of Ones in a Block

The test determines whether the length of the longest run of ones is the same as expected in a random sequence within *M*-bit blocks, after dividing the whole binary sequence into some parts with the length of *M*. As a matter of fact, after dividing the sequence into *M*-bit blocks, the frequencies of the longest runs of ones in each block should be computed. In this test, only the run of ones is necessary to check, because the irregularity in the expected length of run of ones implies that, there would be an irregularity in the length of run of zeros [14].

E. Binary Matrix Rank Test

By focusing on the rank of disjoint sub-matrices of the whole sequence, it checks for the linear dependencies between fixed length substrings of the sequence. In this regard, the sequence is divided into *R* x *C* –bit disjoint blocks (*R* denotes the number of rows, and *C* indicates the number of columns in each matrix), in the way that each row is filled with *C*-bit blocks of the sequence. The number of blocks would be the result of the division of the length of the sequence, and the multiplication of *R* and *C* [14]. This test is mentioned in Diehard battery test as well [15].

F. Discrete Fourier Transform (Spectral) Test

The purpose of the test is to find the periodic features, and detect the patterns which are repeated near each other. By focusing on the peak heights in the discrete Fourier transform of the sequence, this test defines the deviation of randomness by computing the number of peaks, to determine whether exceeding a threshold. In February 2009, NIST announced that they have found a problem in Discrete Fourier Transform Test and they advised to disregard the result of this test.

G. Non-overlapping Template Matching Test

This is to define and detect generators which produce too many occurrences of a given non-periodic pattern, or pre-specified target string. An *m*-bit window is used to detect a specific *m*-bit pattern, after partitioning the sequence into *M* blocks. If the pattern is found, the window will be reset to the bit after that pattern, and the search will be resumed, otherwise the window slides over by one bit only [14].

H. Overlapping Template Matching Test

A variation of the non-overlapping template matching test, and the purpose of the test is searching for a specific *m*-bit pattern. Unlike the overlapping template matching test, in this test if there is no match and the pattern is not found, the window slides by one bit position [14].

I. Maurer's "Universal Statistical" Test

This test computes the numbers of bits between matching pattern to indicate if the sequence is compressible without any loss of information since the compressible sequences are considered as non-random [14].

J. Linear Complexity Test

The test's focus is on the length of a Linear Feedback Shift Register (LFSR), and its purpose is to determine if the sequence is complex enough in compare to a random sequence. Random sequences are often characterized by the

length of a Linear Feedback Shift Register (LFSR), if LFSR is too short, the sequence will be non-random. Statistically, the test measures how well the observed number of occurrence of LFSRs matches the expected number of which, under an assumption of randomness [14].

K. Serial Test

This is to check the frequency of all possible overlapping *m*-bit patterns, and to define if every *m*-bit pattern has the same chance to appear as every other *m*-bit patterns in the entire sequence. A random sequence has a uniformity like this, the number of occurrence of the 2^m *m*-bit patterns across the whole sequence are required to be the same as expected from random sequence [14].

L. Approximate Entropy Test

Calculating the frequency of all possible *m*-bit patterns, this test is to determine if two physically adjacent overlapping block with the lengths of *m*, and then *m*+1, is the same as expected for a random sequence [14].

M. Cumulative Sums (Cusum) Test

Execution of cumulative sum test on a random sequence of zeros and ones should produce a near zero value. This is done by converting (0, 1) to (-1, 1), and then calculating the cumulative sum of the sequence. A large result shows a non-random sequence [14].

N. Random Excursion Test

Similar to the Cumulative Sum test, this test determines the number of cycles that have exactly *k* visits in a cumulative sum random walk. The cycle includes a sequence of steps of unit length taken periodically at random which begins and returns to the origin. Thus, the main purpose of the test is to define whether the number of visits to a particular state in a cycle is the same as expected from a random sequence [14].

O. Random Excursion Variant Test

The test measures the number of times that a particular state is visited in a random walk, and defines the deviations from the same in a random sequence, and detects these deviations from expected number of visits to various states in a random walk [14].

III. HOPFIELD NEURAL NETWORKS

A recurrent neural network, the Hopfield model was proposed in 1982 [11]. The learning in a Hopfield network is done by means of a weight adjustment mechanism that directly relates to minimization of an energy function that decreases over time in each iteration and finally stabilizes in some point of the state space representing the problem. The basic idea of Hopfield Neural Networks (HNN) is to memorize some patterns as stable points by associating them with specific inputs to the network. That is, after some iteration the network goes to stable points in the state space that relates to the pattern that is memorized. Stability, therefore, is the most important features of Hopfield neural networks [15]-[19].

The ability to converge in Hopfield networks is strongly related to network architecture, network initial condition, and updating rule mode. The convergence of the network occurs when the weight matrix is symmetric. Thus, there might be some alternatives which cause the network not to converge, e.g., by (i) applying an initial asymmetric weight matrix consists of large positive numbers in diagonal, (ii) letting two or more neurons active simultaneously, and (iii) using large network and training it with orthogonal and uncorrelated patterns [5].

In this paper, we develop a HNN with following conditions to guarantee non-convergence:

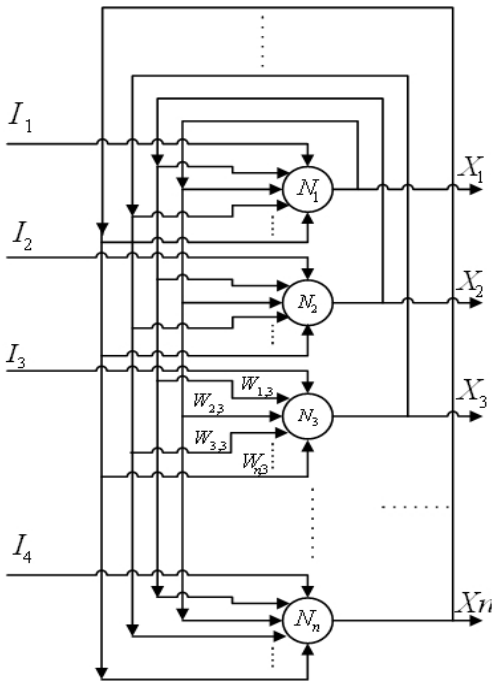

Fig. 1. The structure of our HRNN, which each neuron connects to itself as well

- The structure of the proposed HNN is fully connected, that is, self-feedback exists (Fig. 1),
- We use a nonlinear function, tanh(*x*) where *x* is summation of all inputs of a neuron, as the activation function for our neurons,
- Weights matrix of our HNN is asymmetric where the upper triangle of weight matrix contains positive numbers and lower triangle of matrix contains negative numbers,

TABLE I. THE RESULT OF THE NIST PRNG TEST FOR 4 DIGIT SELECTION HNN PRNG

PRNG Test	1-digit selection HNN Experiment 1		1-digit selection HNN Experiment 2		1-digit selection HNN Experiment 3		3-digit selection HNN Experiment 1		3-digit selection HNN Experiment 2		3-digit selection HNN Experiment 3	
	P-Value	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*
Frequency	0.0589	0.91*	0.0066	0.88*	0.0126	0.90*	0.6371	0.96	0.5749	0.95*	0.4190	0.94*
Block Frequency	0.2896	0.97	0.0109	0.95*	0.7597	0.98	0.3838	0.96	0.2368	0.99	0.9114	0.96
Cumulative Sums-Forward	0.1453	0.91*	0.0004	0.88*	0.0428	0.89*	0.9114	0.95*	0.5749	0.95*	0.3838	0.94*
Cumulative Sums-Reverse	0.3504	0.93*	0.0029	0.89*	0.0155	0.88*	0.0006	0.95*	0.6993	0.94*	0.0219	0.94*
Runs	0.0269	0.91*	0.0909	0.92*	0.0308	0.88*	0.1372	0.97	0.4749	0.97	0.6786	0.94*
Longest Run	0.4559	0.89*	0.3345	0.91*	0.0909	0.89*	0.2022	0.94*	0.5749	0.96	0.9942	0.96
Rank	0.4372	0.95*	0.4559	0.97	0.0000	0.93*	0.6786	0.99	0.2896	1	0.4943	0.98
Non Overlapping Template	0.3102	0.92*	0.2485	0.92*	0.1902	0.90*	0.4225	0.96	0.4199	0.95*	0.3669	0.94*
Overlapping Template	0.0487	0.89*	0.0251	0.92*	0.0308	0.88*	0.4190	0.97	0.0456	0.95*	0.8343	0.94*
Universal	0.0456	0.89*	0.0757	0.91*	0.1025	0.90*	0.8343	0.97	0.6163	0.97	0.1223	0.95*
Approximate Entropy	0.1223	0.90*	0.0004	0.85*	0.0022	0.87*	0.4559	0.93*	0.0519	0.95*	0.0057	0.92*
Random Excursions	0.3295	0.99	0.2643	0.99	0.6490	0.98	0.2899	0.98	0.2669	0.99	0.5122	0.99
Random Excursions Variant	0.3413	0.99	0.4265	0.99	0.5165	0.98	0.3648	0.99	0.3195	0.99	0.4968	0.99
Serial(m=5)	0.0647	0.06*	0.1504	0.91*	0.0057	0.87*	0.3504	0.97	0.4144	0.93*	0.2985	0.94*
Linear Complexity	0.7399	0.94*	0.2368	0.94*	0.2896	0.92*	0.4943	0.97	0.8977	1	0.4559	0.99

- The diagonal of weight matrix contains large positive numbers,
- We use large number (100) of neurons.
- In selecting the weights if the output of one neuron is close to or more than 1 then in the next iteration that neuron amplifies itself by the weight of the corresponding branch regardless of other inputs of the neuron. This is also valid for output close to or less than -1 with decreasing impact on the neuron. In other word, this makes the neuron to always fire with ±1 or bigger than ±1 in all iterations and accordingly amplifying itself. To avoid this, we set a condition that the summation of all the weights for inputs to each neuron must be less than 1 and greater than -1.
- The output of each neuron in each iteration is calculated by (11):

$$X_i^{new} = \mathrm{sgn}\left(\sum_{j=1}^{n} W_{j,i} X_j - \theta_i + I_i \right) \qquad (11)$$

  Where I, X, W, θ, and n signify input, output, weight, threshold and number of neurons respectively.
- In our HNN θ is zero and in the first iteration I is 1. We also need nonlinear activation function so we use tanh(x) function instead of sgn(x) function.

Convergence in a HNN is achieved when the corresponding outputs of all neurons reach a stable state or oscillate between a limited number of states [19]. The definition of state stability, however, directly relates to the degree of accuracy of our calculation. For example, if a state is stable with an accuracy of n digits after the decimal point, the very same state might not be stable for an accuracy of $m>n$ digits after decimal point. While digits with higher order of significance have converged, the other digits with lower order of significance have not converged yet, and may converge in the following iterations. Therefore, any stable neuron in a HNN may be viewed as unstable if the accuracy of the calculation is increased. This holds true for both cases of stability where there is (i) on stable point, or (ii) a limited number of stable points.

IV. HNN AS PRNG

We implemented a PRNG based HNN with 100 neurons with HNN weights playing the role of PRNG seed. To reduce the convergence possibility, we selected 10,000 weights for the proposed networks (weight selection explained in section III). Furthermore, by considering 15 digit precision we both increased the accuracy of our calculation and decreased the chance of convergence of the network. We also devised a sampling mechanism whereby random numbers can be acquired from the outputs of network. This was done by removing the 15th digit in output of each neuron, and extracting digits from the end. If the extracted digit is between 0 to 7 then we convert it to a 3-bit binary format and if it is 8 or 9 we discard it. A collection of such 3-bit binary numbers is used as a Random Sequence of bits. We set α parameter to 0.01 and the sequence length to 1,000,000 in our test module. The minimum proportion pass rate for each statistical test is approximately 0.96015 based on NIST PRNG Test Module. We repeat the experiment for 3 times and the results are tabulated in Table I, 1-digit selection columns. In Table I, (*) indicates that our HNN PRNG has not passed its corresponding test. In Table I, *P-Value* is the result of the test and *Prop* is proportion for 100 sequences.

By investigating 1-digit selection column of table I, we observed that in most of the tests e.g. Frequency and Approximate Entropy tests, P-Values do not indicate good result. Moreover our HNN PRNG could not pass most of the tests like Cumulative Sums-Forward/Reverse, Runs, Rank and Longest Run tests according to the proportion results of those tests. The obtained results imply that there might be some kind of patterns among the digits in output of neurons in some iteration, i.e. by selecting one digit from the output of each neuron in HNN PRNG, the final bit Sequence does not show good statistical characteristic to pass the PRNG Test.

TABLE II. THE RESULT OF THE NIST PRNG TEST FOR 5 & 7 DIGIT SELECTION HNN PRNG

PRNG Test	5-digit selection HNN Experiment 1		5-digit selection HNN Experiment 2		5-digit selection HNN Experiment 3		7-digit selection HNN Experiment 1		7-digit selection HNN Experiment 2		7-digit selection HNN Experiment 3	
	P-Value	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*
Frequency	0.2757	0.95*	0.0308	0.96	0.0009	0.96	0.5955	0.97	0.0117	0.96	0.4011	0.96
Block Frequency	0.5544	0.93*	0.6993	0.98	0.1153	0.97	0.4190	0.99	0.7399	0.97	0.4943	0.96
Cumulative Sums-Forward	0.3669	0.96	0.0308	0.97	0.0191	0.95*	0.2622	0.97	0.0401	0.97	0.3669	0.98
Cumulative Sums-Reverse	0.0167	0.94*	0.1372	0.96	0.0046	0.95*	0.3041	0.97	0.0109	0.96	0.0805	0.96
Runs	0.1025	0.91*	0.8513	0.96	0.0082	0.93*	0.2492	0.98	0.0179	0.98	0.7197	0.98
Longest Run	0.1916	0.96	0.7791	0.95*	0.0308	0.95*	0.8343	0.99	0.5955	1	0.3041	0.98
Rank	0.9463	0.97	0.9357	1	0.4749	0.98	0.0487	0.99	0.7981	0.99	0.8343	1
Non Overlapping Template	0.1913	0.92*	0.4029	0.95	0.3615	0.94*	0.5044	0.98	0.5108	0.98	0.4818	0.96
Overlapping Template	0.3669	0.92*	0.1916	0.96	0.5341	0.96	0.5141	0.99	0.4190	0.97	0.7197	0.96
Universal	0.3669	0.97	0.5341	0.96	0.1916	0.94*	0.6371	1	0.2896	0.98	0.1815	0.99
Approximate Entropy	0.0219	0.87*	0.9114	0.94*	0.0082	0.92*	0.2896	0.96	0.0965	0.98	0.3190	0.96
Random Excursions	0.6230	0.98	0.2211	0.98	0.5006	0.99	0.4499	0.99	0.4710	0.98	0.4682	0.99
Random Excursions Variant	0.4189	0.99	0.2045	0.99	0.4089	0.99	0.4638	0.99	0.5086	0.99	0.4497	0.99
Serial(m=5)	0.0024	0.88*	0.4206	0.96	0.0699	0.92*	0.1662	0.95*	0.0537	0.97	0.3981	0.94*
Linear Complexity	0.2133	0.99	0.0219	0.99	0.6371	0.97	0.3838	1	0.8343	0.97	0.3838	1

TABLE III. THE RESULT OF THE NIST PRNG TEST FOR 9 DIGIT SELECTION HNN PRNG

PRNG Test	9-digit selection HNN Experiment 1		9-digit selection HNN Experiment 2		9-digit selection HNN Experiment 3	
	P-Value	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*
Frequency	0.6163	0.97	0.5141	0.95*	0.7399	0.99
Block Frequency	0.3190	0.98	0.2622	1	0.2622	1
Cumulative Sums-Forward	0.0329	0.96	0.9463	0.95*	0.9114	0.99
Cumulative Sums-Reverse	0.6786	0.95*	0.0269	0.96	0.4372	0.99
Runs	0.4943	0.98	0.3838	0.94*	0.2022	0.97
Longest Run	0.5544	1	0.2896	0.98	0.4011	1
Rank	0.5749	1	0.1537	1	0.1153	1
Non Overlapping Template	0.4142	0.96	0.4476	0.96	0.5004	0.98
Overlapping Template	0.5544	0.99	0.2757	0.97	0.9114	0.99
Universal	0.6579	0.98	0.1153	1	0.7197	0.97
Approximate Entropy	0.1372	0.92*	0.3190	0.92*	0.7791	0.97
Random Excursions	0.4682	0.98	0.3344	0.97	0.3656	0.98
Random Excursions Variant	0.2955	0.98	0.3274	0.99	0.3912	0.97
Serial(m=5)	0.4127	0.92*	0.4404	0.955*	0.7357	0.975
Linear Complexity	0.5341	0.99	0.8343	1	0.1453	0.99

The question that we will investigate is, "What is the influence of increasing the number of digits to be selected from output of each neuron on the PRNG test results?"

V. EFFECT OF DIGIT SAMPLING MECHANISM ON HNN AS PRNG

In 1-digit selection column of Table I, we noticed that in most of the test, our HNN PRNG does not provide good P-Values and Proportions. NIST test is a statistical test hence if any kind of pattern or irregularity exists in random sequence numbers, it would try to find it. 1-digit selection columns of Table I signifies that produced random sequences do not have good statistical characteristic to pass PRNG Tests.

To perform new experiments, we increased the number of extracted digits from the output of each neuron, consequently we could examine the influence of number of extracted digit from the output of each neuron in PRNG Test results.

We conducted several system testing with different number of digits extracted from output of each neuron. As mentioned, the outputs of the neurons are numbers with 15 digits after decimal. We discarded the last digit and picked the number of digits we wanted from the end to produce the random sequence. In this experiment we tested our HNN PRNG with 3, 5, 7 and 9 digits extracted from the output of our HNN and repeated the experiment 3 times for each case. The results are shown in tables I, II and III – (*) indicates that our HNN PRNG has not passed corresponding test.

In most of the test like Frequency Test, Runs Test and Longest Run test, P-values in 3-digit selection columns are bigger in comparison with their corresponding in 1-digit selection columns - Table I. Some tests such as Linear Complexity that did not pass 1-digit selection passed 3-digit selection. However, 3-digit selection test of HNN PRNG could not pass some tests like Cumulative Sums-Forward and Cumulative Sums-Reverse that 1-digit selection was unable to pass.

Experiment 2 of 5-digit selection column of table II, has passed all the tests except Approximate Entropy and Serial test. It seems we are moving toward a better PRNG, by increasing the number of extracted digits. Be noticed that in experiments 1 and 3 in 5-digit selection test PRNG could not pass some tests. However, not the same tests are failed in the two experiments.

Noticeably, in Table II, Experiment 2 of 7-digit selection tests, shows HNN PRNG has passed all tests with higher P-values and Proportions than previous tests. However in experiments 1 and 3, HNN PRNG failed Serial Test with a close number to pass/fail threshold of proportion result.

HNN PRNG, with respect to Table III, experiment 3 of 9-digit selection tests, has also passed all the tests, but in experiment 1 and 2 some tests are failed. P-Values in 9-digit selection experiments in comparison with 7-digit Selection columns of Table II, shows increase. Due to a great chance of convergence, we did not conduct 10- or 11-digit selection tests.

VI. CONCLUSION

Pseudo random number generators with good quality of randomness are essential to all cryptographic applications. Inspired by the convergence problem of Hopfield Neural Networks, we examined using HNN as PRNG.

We designed and developed a non-converging HNN based on which we produced and then evaluated random sequence number using NIST PRNG Random test package. We showed that number of digits that we extract from output of each neuron has a big influence in performance of our HNN PRNG in terms of passing the PRNG tests. We showed that by increasing the number of digits extracted from the output of each neuron in HNN PRNG a better PRNG is achieved.

REFERENCES

[1] NIST, "Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules," July 2009 Available: http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf.

[2] R. Gennaro, "Randomness in cryptography," IEEE Journal of Security & Privacy, vol. 4, no. 2, pp. 64-67, April 2006.

[3] F. Pareschi, R. Rovatti, and G. Setti, "Second-level NIST Randomness Tests for Improving Test Reliability," in Proc. of the IEEE International Symposium on Circuits and Systems, pp. 1437-1440, May 2007.

[4] L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," SIAM Journal on Computing, vol. 15, pp. 364–383, May 1986.

[5] T. Schmidt, H. Rahnama and A. Sadeghian, "A Review of Applications of Artificial Neural Networks in Cryptosystems," in Proc. of the International Symposium on Soft Computing for Industry (ISSCI 2008).

[6] M. Arvandi, S. Wu, and A. Sadeghian, "On the Use of Recurrent Neural Networks to Design Symmetric Ciphers," IEEE Computational Intelligence Magazine, vol. 3, no. 2, pp. 42-53, May 2008.

[7] I. Woungang, A. Sadeghian, S. Wu, S. Misra, and M. Arvandi, "Wireless Web Security Using a Neural Network-Based Cipher," Chapter II in Web Services Security and E-Business, G. Radhammani and G. S.V. Radha Krishna Rao (Eds.), Idea Group Publishing Inc., USA, ISBN: 1-59904-168-5, pp. 32-56, 2006.

[8] D. A. Karras and V. Zorkadis, "Overfitting in multilayer perceptron as a mechanism for (pseudo) random number generation in the design of secure electronic commerce systems," in Proc. of the Information Systems for Enhanced Public Safety and Security, IEEE/AFCEA, EUROCOMM 2000.

[9] Y. Wang, Z. Shen and H. Zhang, "Pseudo Random Number Generator Based on Hopfield Neural Network", in Proc. of the Fifth International Conference on Machine Learning and Cybernetics, pp. 2810-2813, August 2006

[10] Y. Wang, G. Wang and H. Zhang, "Random Number Generator Based on Hopfield Neural Network and SHA-2 (512)," Advancing Computing, Communication, Control and Management, Springer Berlin Heidelberg, vol. 56, pp. 198-205, December 2009.

[11] S. Haykin. Neural Networks: A Comprehensive Foundation. 3rd edition, Prentice Hall, 2009

[12] M. Geisler, M. Krøigård and A. Danielsen, "About Random Bits," December 2004, Available: http://www.cecm.sfu.ca/~monaganm/teaching/CryptographyF08/random-bits.pdf

[13] G. Marsaglia and W. W. Tsang, "Some difficult-to-pass tests of randomness," Journal of Statistical Software, vol. 7, no. 3, pp. 1-8, Jan 2002.

[14] NIST, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," August 2008, Available: http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf

[15] G. Marsaglia, "DIEHARD Statistical Tests," Available: http://www.stat.fsu.edu/pub/diehard/.

[16] J. J. Hopfield, "Neural Network & Physical Systems with Emergent Collective Computational Abilities," in Proc. of the Natl. Acad. Sci, vol. 29, pp. 2554–2558, Apr 1982.

[17] M. B. Menhaj and N. Seifipour, "A new implementation of discrete-time Hopfield net with higher capacity and speed of convergence," in Proc. of the International Joint Conference on Neural Networks, pp. 436-441 vol. 1, 2001.

[18] R. Ma, P. Chu and S. Zhan, "Stability Conditions for Discrete Delayed Hopfield Neural Networks," in Proc. of the Third International Conference on Natural Computation, vol. 1, pp. 468-472, August 2007.

[19] R. Ma, Y. Xie, S. Zhang and W. Liu "Convergence of discrete delayed Hopfield neural networks," Computers & Mathematics with Applications, vol. 57, no. 11-12, pp. 1869-1876, June 2009.