# Pseudo Random Number Generator Using Elman Neural Network

V.V.Desai, V.B.Deshmukh
Research Centre,Dept.of Electronics and Communication Enggineering
Gogte Institute of Technology
Belgaum,Karnataka,India
veenades@gmail.com, veenadeshm@yahoo.com

D. H. Rao
Dept. of Electronics and communication Engineering
Jain College of Enggineering
Belgaum,Karnataka,India
dr.raodh@gmail.com

***Abstract*-This paper proposes a pseudo random number generator using Elman neural network. The proposed neural network is a recurrent neural network able to generate pseudo-random numbers from the weight matrices obtained from the layer weights of the Elman network. The proposed method is not computationally demanding and is easy to implement for varying bit sequences. The random numbers generated using our method have been subjected to frequency test and ENT test program. The results show that recurrent neural networks can be used as a pseudo random number generator(prng).**

***Keywords: deterministic algorithm, Elman network, prng, recurrent neural network.***

## I. INTRODUCTION

Random number generation is important in many scientific contexts, from physical and statistical simulation to cryptography and software testing. Pseudo-random number generators are intended to be general-purpose vehicles for the creation of random data used in these areas [1][2][3]. Due to the deterministic nature of computers and the common need to be able to repeat sequences of pseudo-random data, pseudo-random number generators are based on deterministic algorithms.

Many different methods exist for generating pseudo-random numbers like Blum-Blum-Shub[4], Mersenne Twister algorithms[5][6] etc. Random numbers can be generated using neural networks as neural networks are highly non-linear mathematical systems [7][8][9].

The authors of [10] use the dynamics of neural networks with random orthogonal weight matrices to generate random numbers. Neuronal plasticity is used in [11] to generate pseudo-random numbers.

In this paper the Elman recurrent neural network is used to generate random numbers. The Elman network differs from conventional two-layer networks in that the first layer has a recurrent connection. The delay in this connection stores values from the previous time step, which can be used in the current time step.

In section 2 of the paper the background for prng generation and neural networks is provided. Section 3 gives the implementation details. In section 4 we discuss the algorithm. The results obtained by applying the ENT test to the prng's generated by the Elman network are presented in section 5. We present our conclusions in section 6.

## II. BACKGROUND

The construction of useful, efficient, and effective prng's has motivated research in computer science, applied mathematics, and other related fields since the use of systems and computers for statistical computation began. Additionally, the field of cryptography relies heavily on the use of prng's, especially for data encryption[12][13]. In this context, prng's must be secure and unpredictable, but they must also be efficient[14][15][16].It is easy to see the main conflict of random number generation when viewed in the context of cryptography: there is almost always a trade-off between efficiency (i.e., execution time) and effectiveness (i.e., level of security). Many good prng's exist today, such as the Blum-Blum-Shub and Mersenne Twister algorithms.

However, they are often poor choices for cryptographic applications because of their high predictability. On the other hand, this is often a positive factor that contributes to their use in simulations, since it is possible to "jump to" the state of a given linear system at any discrete time. By implication, a particular (sub) sequence can be regenerated arbitrarily without having to simulate the entire sequence again from the beginning. Some non-linear systems, on the other hand, do not possess this characteristic, and are therefore better suited for cryptography. Their dynamics are often highly dependent on given variable, such as time.

One area of random number generation that has previously seen little research is the use of neural networks to create pseudo-random numbers. Neural networks are highly non-linear, mathematical systems that are meant to simulate, in some abstract sense, the dynamics of neurons in the brain (more specifically the neocortex). Neural network research began by using networks with little similarity to biological systems, but more recent research has begun to take advantage not only of the functionality of neurons, but also their structure in the brain. External forces (i.e., not inherent to the neuron itself) have also become an integral part of neural network research, and represent a fundamental aspect of our experiments. Some attempts have been made to utilize the dynamics of neural networks with random orthogonal weight matrices for random number generation. These networks were successful in generating statistically random numbers that

978-1-4244-9477-4/11/$26.00 ©2011 IEEE

pass industry-standard tests with a success rate similar to that of other well established random number generators. The neural networks presented in this paper provide examples of non-linear systems.

## III. IMPLEMENTATION

The Elman network commonly is a two layer network with feedback from the first-layer output to the first layer input. This recurrent connection allows the Elman network to both detect and generate time varying patterns.

The Elman network used in this paper is designed using Matlab code. Even if two Elman networks as shown in Fig.1, with the same weights and biases, are given identical inputs at a given time step, their outputs can be different because of different feedback states.

Because the network can store information for future reference, it is able to learn temporal patterns as well as spatial patterns. The Elman network can be trained to respond to, and to generate, both kinds of patterns. The networks were first trained with 5 neurons and gradually increased to 2000 neurons. The number of epochs for the training given in the results of Table.1 is 50. For larger networks, increase in the number of epochs leads to computational problems.

In this paper, the initial weight matrices of Elman neural network *LW* as in Fig.1 are trained to take advantage of the behavior of input keyword to shape the dynamics of neural networks. Their values after training are obtained as the pseudo random sequences. The algorithm for obtaining the pseudo random sequence is given in the next section.

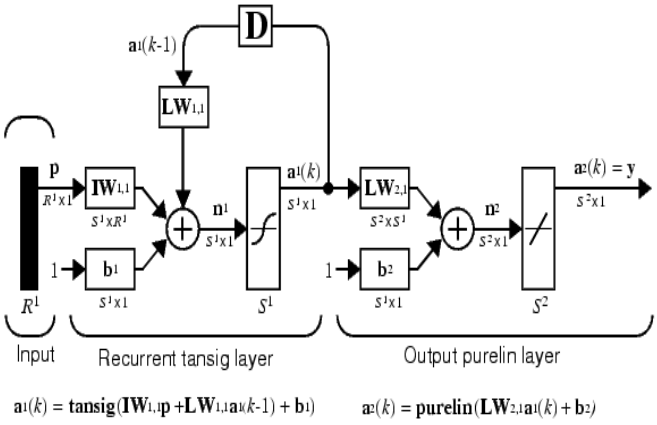


Figure 1. Detailed diagram of a Elman Network

## IV. ALGORITHM:

- Use an input keyword 'keyword' to generate a unique number sequence p and target sequence q for the Elman neural network .

  *Pseq=f(t,p);*

  *qseq=f(t,q);*

  An Elman network is generated for the required number of neurons n and p and q :

  *elmannetwork(n,,p,q)*

- The generated Elman network is trained for the input sequence pseq and target sequence qseq

  *trainelmannetwork(pseq,qseq)*

- The initial layer weight matrix [$LW_{\{1,1\}}$] *LWmat* is used for generating the pseudo random number.

- Derive *newLWmat =abs(LWmat)-meanof(LWmat)*

  prng =reshape (newLWmat)

The randomness of the generated sequence is tested using standard software test. There are several popular test packages like DIEHARD, NIST testsuite, CRYPTX and ENT.They work similar, and usually include common tests such as frequency test, long run test, pattern test, correlation test, and so on. These tests assign a property to the sequence, and then test the sequence, comparing with the properties of a sequence that don't have the assumed property. To test the randomness of data generated in this paper, ENT test program has been used. Additionally the frequency test has been done which gives the count of 1's and 0's generated.The choice of ENT was made as it is simple , can be used to test small sequences and provides almost all the tests provided in DIEHARD and NIST.

## V. RESULTS

The neural networks used in this paper are simple, easy to implement and faster than other generators. We have used ENT pseudo-random number sequence test program on the generated sequences. For each test, the ENT test suite program generates entropy, chi-square value, arithmetic mean value, Monte Carlo value for Pi and finally Serial Correlation Coefficient.

The Monte Carlo value for Pi equals 4.0000 for the data generated, whereas for the standard generators it is equal to 3.169834647.

It is well known that all prng's in spite of good performance on statistical test suites possess some weakness. In this regard the neural network used in this paper possesses potential advantages.

ENT pseudo-random sequence test program output for a standard generator input (radioactive decay) file is given below.

- Entropy = 7.980627 bits per character.
- Optimum compression would reduce the size of this 51768 character file by 0 percent.
- Chi square distribution for 51768 samples is 1542.26, and randomly would exceed this value less than 0.01 percent of the times.
- Arithmetic mean value of data bytes is 125.93 (127.5 = random).
- Monte Carlo value for Pi is 3.169834647 (error 0.90 percent).
- Serial correlation coefficient is 0.004249 (totally uncorrelated = 0.0).

The results of the neural network generator implemented for this paper for a network with 500 neurons generating 1000 binary bits are as below

- Entropy = 0.85526 bits per bit.
- Optimum compression would reduce the size of this 168 bit file by 14 percent.
- Chi square distribution for 1000 samples is 3103.3, and randomly would exceed this value less than 0.01 percent of the times.
- Arithmetic mean value of data bytes is 0.2799 (0.5 = random).
- Monte Carlo value for Pi is 4.00000000 (error 27.32 percent).
- Serial correlation coefficient is 0.0783 (totally uncorrelated = 0.0).
- Number of 1's in the sequence of 1000 bits is 477 (balanced output condition).

The plots for the standard generator (used by ENT ) and Elman neural network generated sequences are shown from Fig. 2 to Fig. 6 for arithmetic mean, entropy, Monte Carlo value of pi and serial correlation coefficient parameters as listed in Table 1.

TABLE I. COMPARISON OF RESULTS WITH VARYING NEURONS ( N)

| **Network size** | **100N** | **200N** | **500N** | **1000N** | **2000N** |
|---|---|---|---|---|---|
| **bits generated** | 200 | 400 | 1000 | 2000 | 4000 |
| **Entropy** | 0.86616 | 0.85183 | 0.85526 | 0.86862 | 0.89051 |
| **Frequency test** | 104 | 211 | 477 | 1112 | 2254 |
| **Optimum compression** | 13% | 15% | 14% | 13% | 10% |
| **Chi-Square value** | 576.56 | 1363 | 3103.3 | 5650.1 | 9466.5 |
| **Arithmetic mean value** | 0.2880 | 0.2694 | 0.2799 | 0.2899 | 0.3707 |
| **Monte Carlo value for Pi** | 4.0000 | 4.0000 | 4.0000 | 4.0000 | 4.0000 |
| **Serial correlation Coefficient** | 0.09901 | 0.05067 | 0.078348 | 0.10464 | 0.15105 |

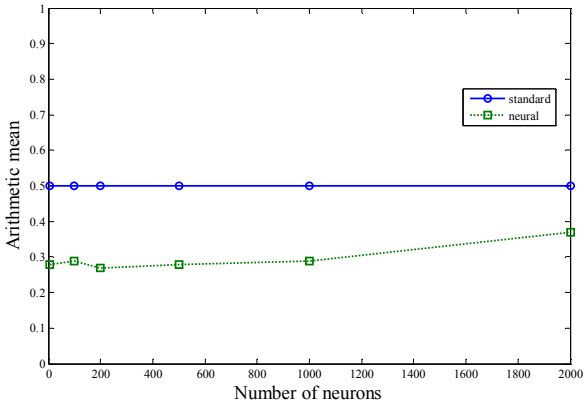


Figure 2. Arithmetic mean Vs No. of neurons

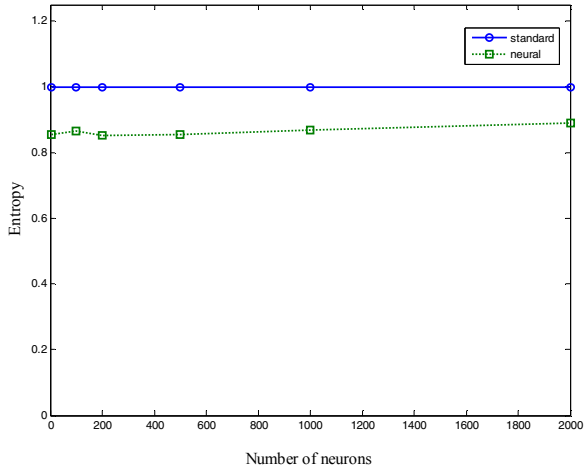


Figure 3. Entropy Vs No. of neurons

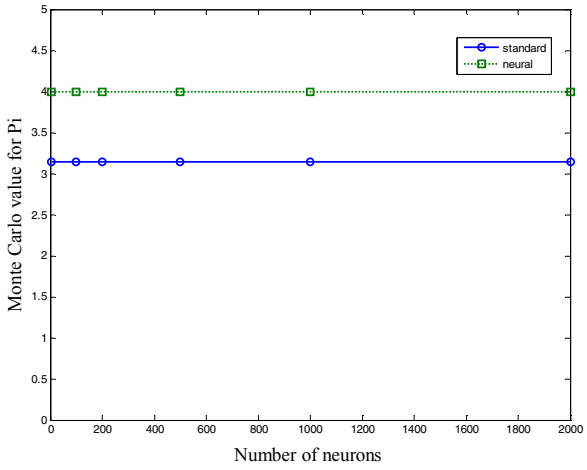


Figure 4. MonteCarlo value pi Vs No. of neurons

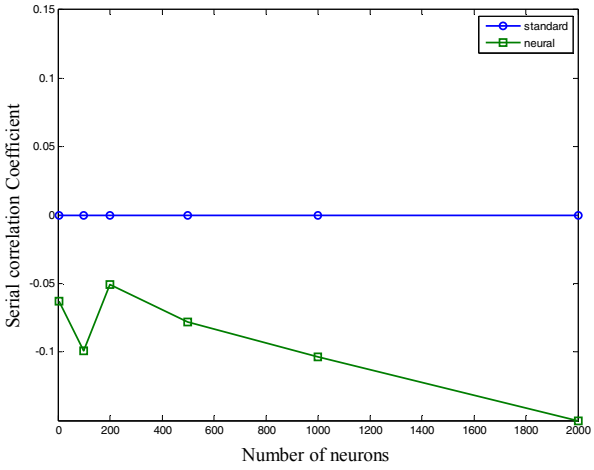


Figure 5. Serial correlation co-efficient Vs No. of neurons

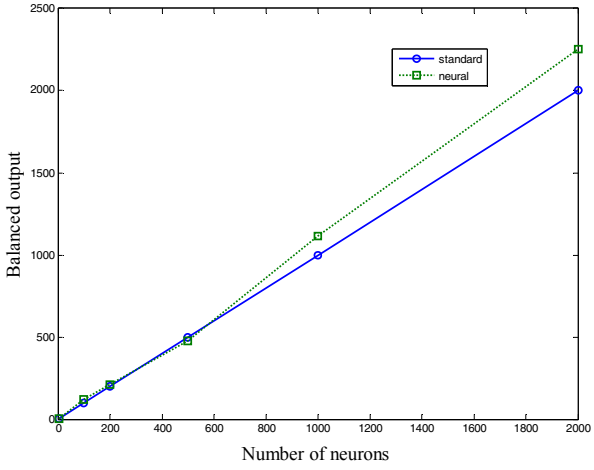


Figure 6. Balanced output Vs No. of neurons

## VI. CONCLUSIONS

From our simulations we conclude that, the neural networks with moderate size of 200 to 1000 neurons produces satisfactory results for Optimum compression, Serial Correlation Coefficient and Frequency test .The Monte Carlo value for Pi remains constant at 4.00000 irrespective of the changes in the size of the network. Optimum Compression is consistent for data as large as 2 million bits. The Arithmetic mean value increases with the increase in the size of the network . So the size of the network should be chosen for optimum values of Arithmetic mean and Frequency test. Because of computational limitations increasing the network size beyond 2000 neurons was not possible.

During the course of simulations, the neural networks implemented in this paper produced random sequences with Monte Carlo value for Pi constantly converging to 4.000. This needs further investigation. Other neural networks with time delay circuits can be experimented. In our further work we propose to use fixed initial weights to initialize the weight matrix to generate cryptographically strong random sequences.

## REFERENCES

[1] G.B. Agnew, "Random Source for Cryptographic Systems," Advances in Cryptology , EUROCRYPT '87 Proceedings, Springer-Verlag, 1988, pp.77-81.

[2] D. Eastlake, S.D. Crocker, and J.I. Schiller, Randomness Requirements for Security, RFC 1750, Internet Engineering Task Force, Dec. 1994.

[3] Pierre L'Ecuyer: Random number generation,In Handbook of Computational Statistics, 2004.

[4] Blum, Blum, and Shub, " A simple unpredictable pseudo random number generator", SIAM Journal on Computing, 15(2):364–383, 1986.

[5] Matsumoto and Nishimura, " Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", ACM Transactions on Modelling and Computer Simulation, 8(1):3–30, 1998.

[6] Abdi, A neural network primer, Journal of Biological Systems, 1994.

[7] Stuart Kauffman, Understanding genetic regulatory networks, International Journal of Astrobiology, 2003.

[8] N.K.Bose,P.Liang,neural Network Fundamentals with graphs,Algorithms and applications ,TATA McGraw-HILL Edition 1998.

[9] Veena V. Desai and D. H. Rao: " S-box design for DES using neural networks for bent function approximation", proceedings of IEEE,INDICON-07,Bangalore,India. .

[10] Yishai M. Elyada and David Horn: " Can dynamic neural filters produce pseudo random sequences?",Artificial Neural Networks: Biological Inspirations – ICANN 2005.

[11] James M. Hughes: "Pseudo-random Number Generation Using Binary Recurrent Neural Networks", A Technical Report submitted to Kalamazoo College 2007.

[12] B. Schneier, Applied Cryptrography, John Wiley & Sons, 1996.

[13] C. Plumb, "Truly Random Numbers, Dr. Dobbs Journal, v. 19, n. 13, Nov 1994, pp. 113-115.

[14] M. Santha and U.V. Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources", Journal of Computer and System Sciences,v. 33, 1986, pp. 75-87.

[15] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic Attacks on Pseudorandom Number Generators",Fast Software Encryption, Fifth International Workshop Proceedings (March 1998), Springer-Verlag, 1998, pp. 168-188

[16] Rukhin, Soto, Nechvatal, Smid, Barker, Leigh, Levenson, Vangel, Banks, Heck- ert, Dray, and Vo: "A statistical test suite for random and pseudorandom num- ber generators for cryptographic applications", NIST Special Publication 800-22,2001.