Can Dynamic Neural Filters Produce Pseudo-Random Sequences?

Yishai M. Elyada[1] and David Horn[2]

[1] Max-Planck Institute of Neurobiology
Department of Systems and Computational Neurobiology
Am Klopferspitz 18, D-82152 Martinsried, Germany elyada@neuro.mpg.de
[2] School of Physics and Astronomy
Raymond & Beverly Sackler Faculty of Exact Sciences
Tel-Aviv University, Tel-Aviv 69978 Israel horn@tau.ac.il

Abstract. Dynamic neural filters (DNFs) are recurrent networks of binary neurons. Under proper conditions of their synaptic matrix they are known to generate exponentially large cycles. We show that choosing the synaptic matrix to be a random orthogonal one, the average cycle length becomes close to that of a random map. We then proceed to investigate the inversion problem and argue that such a DNF could be used to construct a pseudo-random generator. Subjecting this generator's output to a battery of tests we demonstrate that the sequences it generates may indeed be regarded as pseudo-random.

1 Introduction

Dynamic Neural Filters (DNFs) [1] are recurrent networks that transform input space into spatiotemporal behavior. Their dynamics are defined by

$$n_i(t) = H(\sum_j w_{ij} n_j(t-1) - \theta_i) \tag{1}$$

where $n_i(t)$ is the i-th neuron's activation state at time t, H is the Heaviside step function, w_{ij} is the synaptic coupling matrix and θ_i are the neuronal thresholds. This one-step dynamics defines a relation between a state of the system $(n_i)_{i=1\ldots N}$ at time $t-1$ and the state of the i-th neuron at time t.

It is well known that symmetric synaptic matrices lead only to fixed points or two-cycles [2] whereas anti-symmetric ones can lead up to four cycles [3]. The largest cycles may be expected from asymmetric matrices, i.e. ones whose asymmetry $\alpha = \frac{\sum w_{ij} w_{ji}}{\sum w_{ij} w_{ij}}$ is close to zero, and indeed these matrices have been shown to result in cycles that are exponentially long in N, the size of the system [4,5]. Furthermore, it has been shown that the choice

$$\theta_i = \frac{1}{2} \sum_j w_{ij} \tag{2}$$

guarantees that the output of this network will have the largest cycle that a given DNF may possess [1].

In this paper, we show that choosing random orthogonal weight matrices results in significantly longer cycles, the average length of which approaches that of a random mapping. Motivated by this result, we next consider these networks as pseudo-random generators (PRGs) and test the quality of the sequences they generate. Previously known PRGs based on simpler, Hopfield-like networks have been subjected only to partial testing with short generated bit sequences [6]. Other PRG candidates are Cellular Neural Networks [7] however they necessitate a fairly complicated emulation of cellular automata.

2 Optimal Choice of the Weight Matrix

Given the optimal threshold (2) let us prove that, for each neuron, the space of states is naturally divided into two halves: the number of activating states (i.e. the ones whose occurrence at $t-1$ implies $n_i(t)=1$) is equal to the number of inactivating ones. Assume s to be an activating state. Its complement $\bar{s}$, where $\bar{s}_i = 1 - s_i$ will then be inactivating. This follows from the fact that if $\sum_j w_{ij}s_j > \theta_i = \frac{1}{2}\sum_j w_{ij}$ then $\sum_j w_{ij}(1-s_j) < \sum_j w_{ij} - \frac{1}{2}\sum_j w_{ij} = \theta_i$. Obviously the complement of every inactivating state will be an activating one[3] thus completing the proof.

This leads us to suggest imposing orthogonality on the weight matrix by applying the Gram-Schmidt orthogonalization process to N initial row vectors drawn randomly from a Gaussian distribution. The intuition behind this is that it serves to guarantee maximum lack of correlation between outputs of different neurons at any given time. Moreover, since each neuron's hyperplane divides the state-space into two halves in an independent manner, the partitions defined over all the neurons in the system are optimally small; this reduces the probability of closing a cycle in each step, allowing for the production of longer sequences. Numerical testing shows that the largest partitions created by random orthogonal matrices are never larger than about $\frac{2}{3}N$ states.

Figure 1 compares the average length L of cycles as function of N, the total number of neurons, for three different choices of weight matrices: random Gaussian matrices, asymmetrized matrices and random orthogonal matrices. All show exponential increase of the cycle length L with N. Using the parametrization $L = e^{\beta N + \beta_0}$ (choosing $N = 20, 25, 30$ to avoid the finite size effect reported in [5]), we find β values of .215 for the random matrices, .243 for the asymmetrized matrices (compare with [5]), and .320 for the random orthogonal weight matrices. The latter is closer to the exponential coefficient for completely random maps $\frac{\log 2}{2} = .347$. This result is very encouraging given the fact that the DNF is after all a deterministic system.

[3] These statements of complementarity are true for all but a negligible set of matrices where an equality, rather than inequality, can be obtained for some neuron i and some state s. When dealing with integer matrices this can be explicitly avoided by requiring the sum of each row in the weight matrix to be odd.

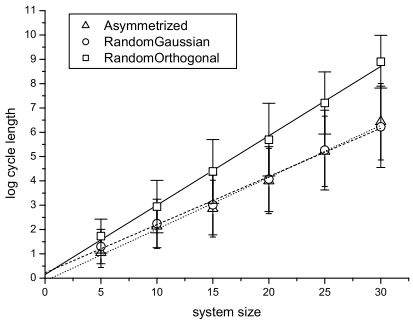

Fig. 1. Dependence of DNF sequence length on N for 3 different methods of generating weight matrices: Averages and standard errors over 100 trials for each system size are shown, with regression lines.

3 Can DNFs Generate Pseudo-Random Sequences?

Pseudo-random generators are functions that expand short, random bit sequences ('seeds') into long pseudo-random bit sequences [8,9] that are computationally indistinguishable from truly random sequences; in other words, the next bit should not be predictable with probability significantly different from $\frac{1}{2}$. PRGs are important primitives in cryptology, and they can be used as a source of effective randomness in the so-called one-time pad cryptosystem (see, e.g., [10]).

PRGs can be constructed using one-way functions (OWFs), by using a hardcore predicate of the OWF (see, e.g., [11] section 3.3.3). Such a construction can be implemented by iteratively applying the function to its own output, and outputting the hardcore predicate at each iteration. For one-to-one length preserving OWFs (bijections of the state-space onto itself), the parity of a certain subset of the bits of the input comprises a hardcore predicate of these functions ([12]).

Is the DNF-step function an OWF? Computing one step of a given DNF dynamics is a simple task, amounting to polynomial time complexity given the digital encoding of a particular DNF. On the other hand, *inverting* one step of the dynamics, i.e. finding a source state for a given target state, is equivalent to the $\mathcal{NP}$-complete problem Integer Programming (IP), finding $x \in \{0,1\}^n$ such that $Wx \geq b$ for some matrix W and vector b. Clearly, the fact that a function is $\mathcal{NP}$-complete does not imply that it is also a OWF, but we will, nevertheless, try to provide evidence for the validity of our conjecture, i.e. that the DNF dynamics function is hard to invert in most cases, qualifying it as a good candidate for a OWF.

To do so, we rely on the well developed field of IP algorithms (see, e.g., [13], [14]). Since these methods were developed to solve general IP problems we use

them as a benchmark for the hardness of our problem. Fig. 2 shows the results of applying the linear programming package GLPK [15] to inverting the step function of DNFs with randomly generated orthogonal weight matrices. The dependence of average solution time on the number of neurons in the system is clearly exponential, testifying to the hardness of solution by this algorithm. We

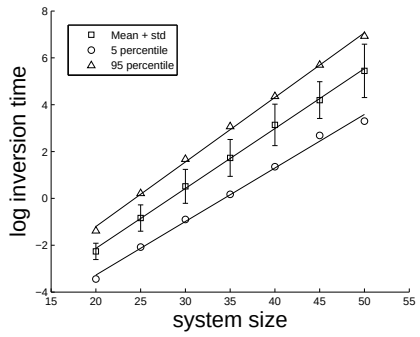

Fig. 2. Dependence of the log inversion time of DNF-step on the size of the system (N). For each N, 100 initial states were randomly chosen for a single random orthogonal DNF and one step was calculated. The mean log time taken by GLPK to find an initial state given the second state is presented as a function of N.

can implement a construct that outputs the parity of a certain subset of bits within the bounds of the DNF model by a DNF 'parity-gadget', which can be added to a given DNF in order to compute the parity of the chosen set of k bits in two steps. The parity gadget consists of a block of k neurons, the i-th of which calculates after one step whether the previous state of the original DNF contained more than i activated neurons, and a parity neuron with a weight vector of $(1, -1, 1, -1...)$ and threshold $\theta = 1$, which uses only inputs from the k-neuron block to calculate the parity of the original DNF two steps previously.

We stress that until we find a weight-matrix construction algorithm that generates preimages of size 1, making the DNF-step function a 1-1 length preserving function, the construct does not necessarily constitute a PRG, even if the DNF-step function is one-way. Nevertheless, we will show some evidence that this construct might still be able to generate pseudo-random sequences, despite this shortcoming.

4 Performance of the DNF Generator

Although our random orthogonal DNFs are not 1-1 functions of the state-space, the DNF generator construct can generate sequences that pass statistical pseudo-randomness tests. We tested this with a comprehensive battery of such tests,

available from NIST (http://csrc.nist.gov/rng/) [16,17] as a means of detecting non-randomness in binary bit sequences generated by pseudo-random generators for cryptographic applications. Generally speaking, consistent failures in any of the 189 listed tests (including multiple variants of several tests) immediately suggests an attack scheme, either explicitly outlined by the test itself, or by the theorem relating unpredictability to indistinguishability from a truly random sequence [8,9]. For each test, the suite generates randomness p-values for all the tested sequences and then uses two final tests to check for deviation from randomness over the set of all 70 bit sequences. The first is a goodness-of-fit test for uniformity of the p-values, and the second is the proportion of sequences that failed each particular test with a 0.01 rejection rate (i.e. $p < 0.01$ for that test).

We used the NIST test suite on 70 1Mbit DNF generator sequences with parity gadgets computing the parity of 5 arbitrary bits. Out of a total of 189 uniformity of p-values and 189 proportion of failure tests, only 2 of the former and 3 of the latter failed. This result is similar to results for other well-tested PRGs such as the Blum-Blum-Shub generator and the RSA generator.

An important point to be made here is that this test suite does not make any use of the information available in the weight matrix. There might still exist, therefore, other ways to 'break' the DNF generator based on this, despite the results shown in Fig. 2. On the other hand, this might also suggest that the DNF generator construct, *without* the weight-matrix being publicly available, is strong enough to generate cryptographically safe pseudo-random sequences.

5 Discussion

Unpredictability is a desired feature of animal behavior in many pursuit-evasion scenarios, with countless examples for such behavior found in nearly all animal groups [18]. Apparently, generating seemingly random sequences of behavior is an innate ability of humans when placed in a competitive situation where unpredictability is an optimal strategy [19], and this ability can also be enhanced by feedback [20,21].

In a presumably deterministic brain, the use of unpredictability as a strategy for survival or evasion must assume a deterministic mechanism of generating unpredictable behavior, in other words, a pseudo-random behavior generator. The model we present suggests that this can be implemented by a simplified neural network without any stochasticity assumptions.

Besides its obvious and straightforward applicability in encryption schemes, the conjectured ability of DNFs to generate pseudo-randomness also holds implications for current-day paradigms of experimental neural research. Namely, partial knowledge of the dynamics of a simple neural system is not always sufficient for effective extrapolation beyond the given data. This is still possible even if the dynamics are fully known for a subset of the neurons and the connectivity of the system is given. Put in another way, recording activity from a few neurons in a large neural system does not guarantee any effective generalization of these data.

References

1. Quenet, B., Horn, D.: The dynamic neural filter: a binary model of spatiotemporal coding. Neural Comput. **15** (2003) 309–329
2. Hertz, J., Krogh, A., Palmer, R.G.: Introduction to the Theory of Neural Computation. Addison-Wesley Longman Publishing (1991)
3. Peretto, P.: An Introduction to the Modeling of Neural Networks. Cambridge University Press (1992)
4. Gutfreund, H., Reger, D.J., Young, A.P.: The nature of attractors in an asymmetric spin glass with deterministic dynamics. J. Phys. A: Math. Gen. **21** (1988) 2775–2797
5. Bastolla, U., Parisi, G.: Attractors in Fully Asymmetric Neural Networks. J. Phys. A: Math. Gen. **30** (1997) 5613–5631
6. Karrasa, D.A., Zorkadis, V.: On neural network techniques in the secure management of communication systems through improving and quality assessing pseudo-random stream generators. Neural Networks **16** (2003) 899–905
7. Crounse, K., Yang, T., Chua, L.O.: Pseudo-random sequence generation using the cnn universal machine with applications to cryptography. Proc. IVth IEEE International Workshop on Cellular Neural Networks and Their Applications (1996) 433–438
8. Yao, A.: Theory and applications of trapdoor functions. Proc. 23th FOCS (1982) 464–479
9. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. Comput. **13** (1984) 850–864
10. Massey, J.L.: An introduction to contemporary cryptology. Proc. of the IEEE **76** (1988) 533–549
11. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2001)
12. Goldreich, O., Levin, L.: Hard-core predicates for any one-way function. Proc. of the 21st ACM STOC (1989) 25–32
13. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley and Sons (1988)
14. Johnson, E., Nemhauser, G., Savelsbergh, M.: Progress in linear programming based branch-and-bound algorithms: An exposition. INFORMS J. Comp. **12** (2000) 2–23
15. Makhorin, A.: Gnu linear programming kit version 4.4. Free Software Foundation (2004)
16. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST (2001) http://csrc.nist.gov/rng/SP800–22b.pdf
17. Soto, J.: Statistical testing of random number generators. Proc. 22nd NISSC (1999)
18. Driver, P.M., Humphries, N.: Protean Behavior: The Biology of Unpredictability. Oxford University Press (1988)
19. Rapoport, A., Budescu, D.V.: Generation of random series in two-person strictly competitive games. J. Exp Psych: Gen. **121** (1992) 352–363
20. Neuringer, A.: Can people behave randomly? the role of feedback. J. Exp Psych: Gen. **115** (1986) 62–75
21. Neuringer, A., Voss, C.: Approximating chaotic behavior. Psych. Sci. **4** (1993) 113–119