Neural network design options for RNG's verification

José Luis Crespo[1], Jaime Gutierrez[1], and Angel Valle[2]

[1] Departamento de Matematica Aplicada y Ciencias de la Computación
Universidad de Cantabria, Santander, Spain
{crespoj,gutierrj}@unican.es
[2] Insituto de Física de Cantabria
Universidad de Cantabria-CSIC, Santander, Spain
valle@ifca.unican.es

Abstract. In this work, we explore neural network design options for discriminating Random Number Generators(RNG), as a complement to existing statistical test suites, being a continuation of the recent paper [22]. Specifically, we consider variations in architecture and data preprocessing. We test their impact on the network's ability to discriminate sequences from a low-quality RNG versus a high-quality one—that is, to discriminate between "optimal" sequence sets and those from the generator under test. When the network fails to distinguish them, the test is passed. For this test to be useful, the network must have real discrimination capabilities. We review several network design possibilities showing significant differences in the obtained results. The best option presented here is convolutional networks working on 5120-byte sequences.

Keywords: Linear Congruential Generator · Linear Congruential Generator on Elliptic Curves · Quantum random number generators · Neural Networks · PyTorch · Statistical test for random number

1 The introduction

Random number generators (RNGs) are widely used in many applications including cryptographycally secured communications, industrial testing, Monte Carlo simulations, massive data processing, quantitative finance, etc. There are two principal methods used to generate random numbers. The first one, called *Pseudorandom Number Generators*(PRNGs), i.e. algorithms which take a small number of bits truly randomly generated, called *the seed*, and expand them to a larger sequence. The second measures some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. In this last class we have Quantum Random Number Generators (QRNGs) that stand out from RNG's because their randomness comes from quantum processes. In this work, we are focusing on laser fluctuations as QRNG.

The quality of RNG is important in various fields including cryptography [13]. The most popular method to experimentally evaluate the fitness of pseudorandom sequences is through the use of the statistical tests such as NIST-STS [11] and DIEHARD [10], see also [14] and [15]. There are also several theoretical measures of pseudorandomness. However they are quite difficult to test in practice because of their high computational complexity. Passing these tests is a necessary condition for the quality of an RNG, but not sufficient, since it has been proven that RNGs considered weak can pass them.

Neural networks are well known machine learning tools that have led to important advances in recent years in many fields, such as image and video object segmentation, medical imaging, face recognition, time series prediction, signal identification, image classification, object detection or human action recognition (see for instance [23] and [24]).

Recently, in several research papers ([16], [17], [18], [19], [20], [21]) neural networks have been suggested as an alternative approach to guarantee the randomness of a given RNG. Those approaches can be considered currently under development, and this work focuses on investigating the possibilities of one of them, which seems the most promising and straightforward to generalize to a test.

Following the idea of [19], the work [22] presented the potential of convolutional networks using the following scheme: ask the neural network to learn to discriminate between two classes of sequences, one coming from an ”optimal” RNG and the other from the RNG to be tested; if the network achieves a success margin significantly different from chance, the RNG to be tested would be considered invalid. For the tests, we will use two RNGs: one weak and one strong, and the network must be able to discriminate between them. The main objective is to explore network design and input preprocessing options to analyze their impact on the network's discriminatory capacity.

The remainder of the paper is structured as follows. We start introducing the main concepts and review the work [22] in Section 2 for later use. Next, in Section 3 we detail the main objectives of the present paper. In Section 4, we show the results achieved with the different explored options. Finally, Section 5 makes some final comments and poses future approaches of research.

2 Preliminaries

Here we review several related results and definitions from [22] for later use and better understanding of the rest of the document.

2.1 Random number generators

The experiment carried out used four kinds of sequences: the laser-based quantum Random Number Generator (the raw sequence and the post-processed one), the binary codification of a large video, the linear congruential generator (LCG), and the linear Congruential Generator on Elliptic Curves (EC-LCG).

Comparison of all of them to the HMAC-DRBG on NIST SP 800-90A(see details in [27]) as the ideal, i.e. high-quality mode of PRNG was performed. An HMAC is a specific type of Message Authentication Code (MAC) involving a cryptographic Hash function and a secret cryptographic key. HMAC-DRBG is a very efficient Deterministic Random Bit Generator (DRBG) that we consider as our Golden Standard Random Number Generator (GSRNG). It has security proofs for a single call to generate pseudorandom numbers and it is backtracking-resistant. On the other hand, it has a machine-verified security proof, that is, the output produced by HMAC-DRBG is indistinguishable from random by a computationally bounded adversary.

Quantum random number generator based on random polarization The Quantum random number generator based on laser fluctuations is presented in [29]. Random numbers are experimentally obtained from the random excitation of the linearly polarized modes of a gain-switched vertical-cavity surface-emitting laser. This randomness is induced by the spontaneous emission that can be considered as quantum noise. Since the raw sequence produced by the QRNG has not passed the NIST test suite [28], we consider the post-processed bit string based on $[n, k, d]$-BCH codes defined over the finite field $GF(2)$ and where $n + 1$ is a power of 2, see [28] and [9].

For the raw input bits $(x_{n-1}, \dots, x_0)$, the output $(y_{k-1}, \dots, y_0)$ is obtained as:

$$\begin{pmatrix} g_{n-k} & \dots\dots\dots & g_0 & 0\dots\dots 0 \\ 0 & g_{n-k} & \dots\dots\dots g_0 & 0\dots\dots 0 \\ \dots & \dots & \dots & \dots \\ 0\dots\dots & 0 & g_{n-k} & \dots\dots g_0 \end{pmatrix} \begin{pmatrix} x_{n-1} \\ x_{n-2} \\ \vdots \\ x_0 \end{pmatrix} = \begin{pmatrix} y_{k-1} \\ y_{k-2} \\ \vdots \\ y_0 \end{pmatrix}$$

and $g(x) = g_{n-k}x^k + \cdots + g_1 x + g_0$ is the cyclic generator polynomial of the $[n, k, d]$-BCH code.

Here we have considered BCH code with parameters $[1023, 1003, 5]$ the generator cyclic polynomial is $x^{20} + x^{15} + x^{13} + x^{12} + x^{11} + x^9 + x^7 + x^6 + x^3 + x^2 + 1$

Linear Congruential Generator Given positive integers a, b and m such that $\gcd(a, m) = 1$ the *Linear Congruential Generator(LCG)* is a sequence x_n of pseudorandom numbers defined by the relation

$$x_{n+1} \equiv (ax_n + b) \bmod m, \qquad n = 0, 1, \dots,$$

where x_0 is the *seed*. Unfortunately the LCG is not suitable for cryptographic purposes, see [2, 8]. Although the author [7] claims that NIST test suites cannot detect the linearity.
In this computational experiment we took the sequences from the rand function in the glibc library version 2-17 without any tunning such that $m = O(2^{32})$ bits, and the output of simple Python LCG code with $m = O(2^{100})$.

Linear Congruential Generator on Elliptic Curves For a prime p, we denote by $\mathbb{F}_p \cong \mathbb{Z}_p$ the field of p elements and, we assume that it is represented by the set $\{0, 1, \dots, p-1\}$.

Let E be an elliptic curve defined over $\mathbb{F}_p$ given by an *affine Weierstrass equation*, which for $\gcd(p, 6) = 1$ takes form $Y^2 = X^3 + aX + b$, for some $a, b \in \mathbb{F}_p$ with $4a^3 + 27b^2 \neq 0$.

We recall that the set $E(\mathbb{F}_p)$ of $\mathbb{F}_p$-rational points forms an abelian group, with the *point at infinity* $\mathcal{O}$ as the neutral element of this group (which does not have affine coordinates).

For a given point $G \in E(\mathbb{F}_p)$ the *Linear Congruential Generator on Elliptic Curves, EC-LCG* is a sequence U_n of pseudorandom numbers defined by the relation

$$U_n = U_{n-1} \oplus G = nG \oplus U_0, \quad n = 1, 2, \dots,$$

where $\oplus$ denote the group operation in $E(\mathbb{F}_p)$ and $U_0 \in E(\mathbb{F}_p)$ is the *initial value* or *seed*. We refer to G as the *composer* of the EC-LCG.

The EC-LCG provides a very attractive alternative to linear and non-linear congruential generators with many applications to cryptography and it has been extensively studied in the literature, see [1, 3–5, 34].

The *.txt* file of 2^{20} bits used was generated running the following SAGEMATH code:

```
 f = open('/Users/PRNG/Desktop/EC_LG.txt', 'a')
size_prime = 512
p=next_prime(ZZ.random_element(2**size_prime))
a=ZZ.random_element(p)
b=ZZ.random_element(p)
if (4*a**3+27*b**2)%p != 0:
    C =EllipticCurve(GF(p),[a,b])
G=C.random_element()
U0=C.random_element()
for i in range(500):
    V=U0+i*G
    f.write(bin(V[0])[2:]+bin(V[1])[2:])
f.close()
```

2.2 Neural networks

Two kind of NN's were used in [22]: LSTM networks and Convolutional ones. Since in this work we aim to compare the performance of convolutional networks, which we analyzed in a previous study, with that of conventional multilayer networks (with full connectivity), we briefly describe these types of models below. For a more detailed description see [25]

LSTM networks A long-term memory network (LSTM) is a specialised recurrent neural network designed to model sequential data. Unlike standard recurrent networks, LSTMs have a single memory cell equipped with gating mechanisms. Within this cell, the input sequence, whether derived from external sources or the preceding layer's output, undergoes intricate filtering via distinct gates, which encompass the integration of the previous cell state. This integration with the previous state imparts dynamic and memory-retentive capabilities to this model. This allows them to capture and retain long-range dependencies and temporal context within sequences, making them well suited for a variety of applications involving sequential data including random sequence prediction and classification. Their ability to capture long-range dependencies and context allows them to identify hidden patterns and temporal relationships in seemingly chaotic data.

LSTM's are also adept at dealing with sequences of varying length, which matches the unpredictability and variability of random sequences. Their adaptive capabilities allow them to process and predict sequences without prior knowledge of their specific characteristics, making them ideal for the downstream application framework presented in a posterior section. In scenarios where traditional models may struggle to provide accurate predictions or classifications due to the inherent irregularities and complexities of RNGs, LSTMs have proven to be indispensable tools showing their ability to encode temporal dependencies and detect subtle patterns.

Convolutional networks (CNN) This type of networks is used where inputs are sequence of same-type values (a row of pixel intensities, a strand of sound pressure values, etc.) Instead of having each processor operate on the whole input array and produce a single output, it calculates a (nonlinear, as above) convolution of the input array with a smaller weight array.

Conventional Fully Connected Networks In addition to the convolutional networks, in this paper we have tested fully connected networks. In all cases, we have used the LeakyRELU as the nonlinear activation, except in the final output, where we used the logistic sigmoid function.

We have tested a network with 200-25-5-1 units in each layer. These sizes are arbitrary; our goal is to be guided by the results to empirically obtain an appropriate size. If the network is too large, there will be a tendency to overfitting, while if it is too small, its accuracy will decrease. Therefore, we will vary the size based on the results. To avoid overfitting, there are other approaches without reducing size, such as: sparse layers ([31]), parameterization of weight matrices (orthogonalization, SVD simplification), weight decay/penalty, and random processor deactivation.

2.3 Procedure

We used the Pytorch package (and PyWavelets for the transformations), running on a GPU with CUDA. In all cases, the procedure is:

1. Select a data set for adjustment, another for control, and another for testing. They are taken from the base set by random selection (using the functions included in Pytorch). For the majority of 256-byte sequences, we verified that sizes above 2^{15} do not make a difference in the results; however, we verified with tests of size 2^{16} or 2^{18}.
2. Adjust the network parameters, using the set selected for this purpose and verifying the error rate on the control set. If there is a consistent increase in the error on this second set, it would indicate overfitting, and the process would be terminated.
3. Analyze the results on the test set. In principle, a response below 0.5 would be considered class 0 (in our case, the strong generator), and above, class 1 (in our case, the weak generator). The 0.5 threshold can be adapted to each case by finding the cutoff point of the response histograms.

To ensure the consistency of the network results and eliminate the influence of the initialization point, we can repeat the last two steps several times. When a good result is the exception rather than the rule, we have considered the test to be poor. The error function to be optimized is the mean square error. The fitting uses accumulated gradients from every few hundred cases (between 100 and 500), using the resilient *back propagation* algorithm [33] most of the time, but in some cases we have obtained better results with Adam [35], with a learning rate of 10^{-3}.

2.4 Implementation and results

The method consisted of training a neural network to differentiate between a PRNG and a truly random sequence, exemplified by the sequences generated by HMAC-DBRG.

The test result is how far can the neural network go in telling apart those two sequence types. We applied the proposed framework to several other RNG's, includes Linear Congruential Generator(LCG), Linear Congruential Generator on Elliptic Curves(EC-LCG), and the laser outputs, raw and post-processed. Including a large enough video file(episode 7 of the continuing education course [12]) as a source of random (meaning unpredictable) sequences.

Table 1 shows the results using LSTM neural networks, including *confusion matrices*:

$$\begin{pmatrix} \boxed{1\,1} & \boxed{1\,0} \\ \boxed{0\,1} & \boxed{0\,0} \end{pmatrix} \qquad \begin{array}{ll} \boxed{1\,1} & RNG\,1\ \ NN\,1 \\ \boxed{0\,1} & HMAC\,0\ \ NN\,1 \end{array} \qquad \begin{array}{ll} \boxed{1\,0} & RNG\,1\ \ NN\,0 \\ \boxed{0\,0} & HMAC\,0\ \ NN\,0 \end{array}$$

Table 1. Results applying the proposed framework to other PRNGs.

PRNG tested	Training size	AO. PRNG	AO. GSRNG	Confusion matrix
VCSEL QRNG	2^{18}	0.47	0.47	$\begin{pmatrix} 50.2\% & 0\% \\ 49.8\% & 0\% \end{pmatrix}$
VCSEL QRNG	2^{19}	0.51	0.51	$\begin{pmatrix} 0.2\% & 49.6\% \\ 0.2\% & 50\% \end{pmatrix}$
Raw QRNG	2^{18}	0.73	0.55	$\begin{pmatrix} 20\% & 30\% \\ 10\% & 40\% \end{pmatrix}$
EC-LCG	2^{18}	0.49	0.49	$\begin{pmatrix} 47.7\% & 2.6\% \\ 47.1\% & 2.6\% \end{pmatrix}$
Video	2^{18}	0.58	0.33	$\begin{pmatrix} 44.1\% & 5.6\% \\ 23.9\% & 26.4\% \end{pmatrix}$
LCG (32 bits)	2^{18}	0.51	0.42	$\begin{pmatrix} 35.5\% & 14.5\% \\ 24.2\% & 25.8\% \end{pmatrix}$
LCG (100 bits)	2^{18}	0.50	0.51	$\begin{pmatrix} 26.9\% & 22.8\% \\ 23.6\% & 26.7\% \end{pmatrix}$

We can see that the VCSEL QRNG passes the test. Not so for the raw QRNG. The elliptic curve generator (EC-LCG) is also successful. We can also see that adding a periodic parameter reset to the LCG is enough to make it pass the test. The *Video* file didn't pass the test, but its performance wasn't that bad; it may rank as good as a naive LCG.

Table 2 shows the effects of several design decision changes, namely: number of processors, network type, layers, sequence length, bytes per element.

Table 2. Performance evaluation of the neural network model with sequential application of hyperparameter or architecture settings.

Design decision	Tested PRNG	Training size	AO. PRNG	AO. GSRNG	Confusion matrix
Increasing processors from 80 up to 150	LCG	2^{15}	0.21	-0.06	$\begin{pmatrix} 50.3\% & 0\% \\ 49.6\% & 0.1\% \end{pmatrix}$
Two layers with 40 and 10 processors	LCG	2^{15}	0.36	0.41	$\begin{pmatrix} 45\% & 5.3\% \\ 39\% & 10.7\% \end{pmatrix}$
Increasing sequence length from 2^8 to 2^9	LCG	2^{15}	0.61	0.23	$\begin{pmatrix} 41.8\% & 8.6\% \\ 11.3\% & 38.3\% \end{pmatrix}$
Increasing sequence length from 2^8 to 2^9	VCSEL QRNG	2^{18}	0.51	0.51	$\begin{pmatrix} 0.5\% & 49.3\% \\ 0.5\% & 49.7\% \end{pmatrix}$
CNN-1	LCG	2^{18}	1	0	$\begin{pmatrix} 50.3\% & 0.2\% \\ 0\% & 49.5\% \end{pmatrix}$
CNN-1	VCSEL QRNG	2^{19}	0.5	0.5	$\begin{pmatrix} 25.1\% & 24.8\% \\ 24.9\% & 25.2\% \end{pmatrix}$
CNN-2 and sequence length=512	VCSEL QRNG	2^{18}	0.5	0.5	$\begin{pmatrix} 29.6\% & 20.3\% \\ 30.1\% & 20\% \end{pmatrix}$
CNN-2 and 2-byte elements	VCSEL QRNG	2^{18}	0.5	0.5	$\begin{pmatrix} 25.3\% & 24.8\% \\ 25.3\% & 24.6\% \end{pmatrix}$

The design decisions that turned effective were: increasing sequence length and using convolutional networks. Results in Table 2 shows that increasing the training set size from 2^{15} to 2^{18} and changing LSTM by CNN improves the capability of the NN to discriminate between those generators and the GSRNG since perfect discrimination is achieved when AO.PRNG and AO.GSRNG are 1 and 0, respectively. VCSEL QRNG was indistinguishable even with the biggest networks that we have tried, as shown in the last three rows of Table 2, since a value of 0.5 for AO.PRNG and AO.GSRNG means that the NN is unable to discriminate between VCSEL QRNG and GSRNG. It remains an open question whether a massively larger network would be able to perform that discrimination.

3 Further improvements

In order to test whether further improvements were achievable, we performed more tests with a fixed generator. We then have two randomly obtained bit series: HMAC and QRNG-VCSEL without postprocessing . We obtain the HMAC bit sequence using the [30] implementation, initializing the generator with 64 bytes obtained from the system's randomness source (Linux) using the $os.urandom$ function, generating 1 byte at a time.

As for the VCSEL with gain-switching generation, it is described in depth in [29], [28], and summarising in Section 2.1.

In both cases, a number of bytes around 2^{20} was generated, which is the base set used in all tests. Each case is one of the sequences, and the expected output is the strong or weak generator label.

Since the generator to be tested is known to be weak, it must be discriminated, see Table 1 and Table 2 of the previous Section 2. Therefore, experiments will be considered successful the higher the discriminatory accuracy of the network; in any case, it must be greater than 50%, which corresponds to chance. As previous section illustrates, we already showed some promising initial results with convolutional networks.

The options we explore here fall into two categories

– Network Design (Conventional Full Connectivity vs. Convolutional)
– Input Sequence Preprocessing

3.1 Preprocessing Options

We take as a starting point the Section 2.4, where the input to the network was the bits obtained by the generator, specifically, sequences of 256 bytes, i.e., 2048 bits.

The possibilities we test here are:

– Fourier Transform
– Wavelet Transform
– Increasing the length of sequences by an order of magnitude, including the possibility of placing them in two dimensions

Fourier Transform In this case, the network's input, instead of being the bits obtained from the random generator, is its Fourier transform. To do this, we take the original sequence and, since they are real, we obtain the transform coefficients with the rfft function from the torch package. The real and imaginary parts of the coefficients, separately, are the two input signals received by the network.

Wavelet Transform This case is analogous to the previous one but using the Wavelet transform. To do this, we take the original sequence and obtain a 4-level decomposition using the wavedec function from the ptwt package. We use the Daubechies-2 basis function with reflection extension. The four series of detail coefficients and the one of approximation coefficients constitute the five input signals to the network. Since the lengths of these sequences are different, we use separate branches of the network for each of them, followed by a common block of dense connectivity layers.

Increasing Sequence Length In this case, maintaining the 1D structure simply involves feeding the network longer sequences. In the case of the 2D structure, we obtained it simply by splitting the 1D sequence into equal fragments and making each fragment a row of a matrix. This matrix constitutes the 2D input to the network. The fragment length we used is the original one from previous analyses, 2048 bits (256 bytes).

4 Results

For clarity, we present confusion matrices in some cases. HMAC (the optimal generator) always appears first, followed by the QRNG without postprocessing (the weak one); we follow the standard approach of indicating the actual labels in rows and the labels assigned by the network in columns.

4.1 Conventional Fully Connected Networks

We tested a network with 200-25-5-1 units in each layer. The accuracy achieved was 63%. Using a validation set, we found a tendency toward overfitting; this led us to believe that the chosen size was too large. We then tried reducing the size to 50-5-1, but it didn't work (accuracy 53%). We found, therefore, that a large size led to overfitting, but a smaller network lost accuracy. We then opted to maintain the large size but use other options to avoid overfitting, which we present below.

We tested automatic pruning on the original network, followed by refitting, but also achieved accuracies of $60 - 63\%$, which did not represent an improvement. Other tests performed included sparse layers ([31]), parameterization of weight matrices (orthogonalization, SVD simplification), weight decay/penalization, and random processor deactivation, but none of these options surpassed the aforementioned accuracy, and therefore, the accuracy of convolutional networks was not achieved.

Although we have not exhaustively explored all layer sizes and numbers, our impression is that the performance is worse than that of convolutional networks.

Therefore, for the remaining tests, we use convolutional networks, as in Section 2.

4.2 Fourier Transform

We arrive at confusion matrices of the type:

$$\begin{pmatrix} 26\% & 24\% \\ 27\% & 24\% \end{pmatrix}$$

We observe that the accuracy obtained is 50%, that is, pure chance, which leads us to abandon this approach.

4.3 Wavelet Transform

A typical histogram of network responses can be seen in Fig.1. It can be seen that approximately half of the QRNG cases are considered good, while the reverse hardly occurs. The confusion matrices obtained in different tests (varying network sizes and data sets) are:

$$\begin{pmatrix} 44 - 45\% & 4.7 - 6.4\% \\ 24 - 26\% & 24 - 26\% \end{pmatrix}$$

which leads to an accuracy of $70 - 71\%$

Although this is an improvement (1% better than the pure-bit option, a small but consistent difference across different tests), it is not large enough to be considered a substantial improvement.

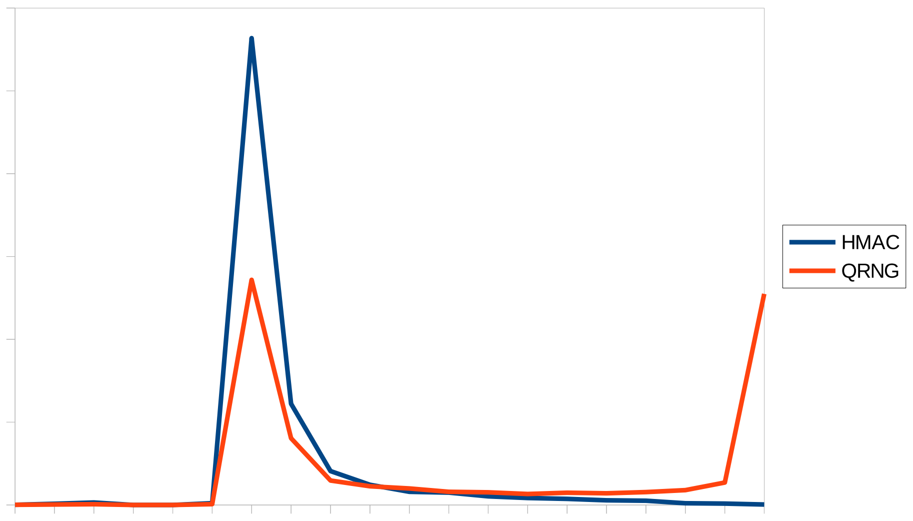

Fig. 1. Histogram with wavelet transformed input

4.4 20-fold increase in sequence length

Using 5120-byte sequences, the network produced responses whose histogram is presented in Fig.2 We see that in exchange for clearly increasing the discrimination of the QRNG sequences, with fewer entering as good, the labeling of the HMACs has more variability, although not enough to mix with the QRNG sequences. The progress over the base model is clear, as can be seen in the following representative confusion matrix:

$$\begin{pmatrix} 48\% & 1.22\% \\ 19.5\% & 31\% \end{pmatrix}$$

The accuracies obtained were in the $75 - 79\%$ range in contrast with an accuracy of $69 - 70\%$ obtained in Section 2.4 using convolutional networks. To achieve accuracies using shortersequences higher than that range, we worked with longer sequences when reaching the final dense connection section. That is, despite having much longer input series, we did not make major reductions in the convolutional section, which means that many more values reach the final conventional section.

If we use the indicator proposed in [19], the average response in case 0 (strong generator) according to the tests is 0.3 ± 0.02, and in case 1 (weak generator) 0.72 ± 0.02.

To see more precisely how the network design affects this case, we present in Table 3 several results indicating the design decision and the quality indicators of the obtained result. The design refers to the convolutional part, where the number of processors in each layer is indicated, separated by hyphens, and, if the sequence is reduced, what the reduction factor is and how it is obtained (maximum or average). The final part is always the same: 3 dense connectivity layers, with 20, 7, and 1 processor. The calculation of

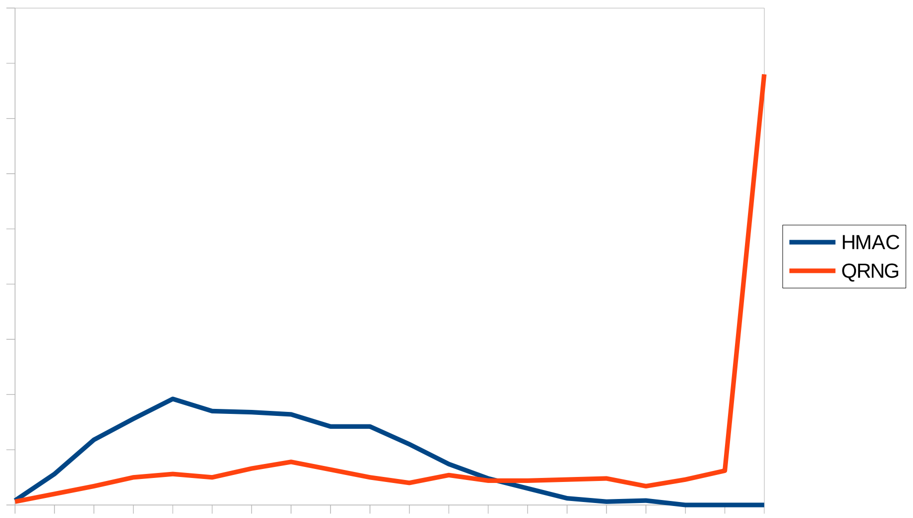

Fig. 2. Histogram with 40960-bit input

the indicators is:

$$S = \frac{\text{class 1 hits}}{\text{total actual class 1 hits}}$$

$$E = \frac{\text{class 0 hits}}{\text{total actual class 0 hits}}$$

$$V = \frac{\text{class 1 hits}}{\text{total indicated by the class 1 network}}$$

Design	Precision	S	E	V
32max2-6max2-5max2-5max2-5	0.76	0.72	0.80	0.80
5max4-6max4-5max4-5max4-5max2-3	0.75	0.63	0.88	0.83
5aver4-6aver4-5aver4-5aver4-5aver2-3	0.76	0.62	0.89	0.86
5aver2-6aver2-5aver4-5-5-3	0.79	0.62	0.96	0.94
5-6-5-5-5-3	0.79	0.61	0.98	0.98

Table 3. Results with various networks on 5120-byte sequences

We tried placing these sequences in a 2D arrangement (20 rows of 2048 bits each), but did not change the precision at all.

5 Conclusions

To summarize the experiments, we have:

- Convolutional networks perform better than dense connectivity networks. We believe this is due to the excess weights, which results in a tendency toward overfitting.
- The Fourier transform is not useful in this context contrary to the wavelet transform, although the latter provides a minimal advantage.
- The most useful factor has proven to be a sharp increase in the length of the sequences. Placing them in two dimensions makes no difference.

As future lines of research, one architecture that we have not dedicated to this task, but that other authors have used for prediction, are networks with an attention mechanism [32], [20]; therefore, we will conduct tests with it to compare it with the convolutional architecture we have used so far. Furthermore, seeing that length increases have the greatest impact, we considered extending the sequence lengths to substantially larger sizes to discern whether the performance of the networks continues to improve.

Acknowledgement

We acknowledge the support from the Advanced Computing and e-Science group at the Institute of Physics of Cantabria (IFCA-CSIC-UC)

A.V. thanks to Ministerio de Ciencia e Innovación, Spain. PID2021-123459OB-C22 MCIN/AEI/FEDER,UE.

References

1. Beelen, P., Doumen, J.: Pseudorandom sequences from elliptic curves. Finite Fields with Applications to Coding Theory, Cryptography and Related Areas, 37-52. Springer-Verlag, Berlin, (2002
2. J. Boyar, ‘Inferring sequences produces by a linear congruential generator missing low–order bits’, *J. Cryptology* **1** (1989) 177–184.
3. J. Gutierrez. “ Attacking the linear congruential generator on elliptic curves via lattice techniques”. Cryptography and Communications volume 14, 505–525 (2022)
4. J. Gutierrez, A. Ibeas . “ Inferring sequences produced by a linear congruential generator on elliptic curves missing high-order bits’. Designs, Codes and Cryptography 45 (2), 199-212
5. S. Hallgren, ’Linear congruential generators over elliptic curves’, *Preprint CS-94-143, Dept. of Comp. Sci.*, Cornegie Mellon Univ., 1994, 1-10.
6. M. Hassan, Cracking Random Number Generators using Machine Learning – Part 1: Xorshift128. https://research.nccgroup.com/2021/10/15/cracking-random-number-generators-using-machine-learning-part-1-xorshift128/
7. S. Hirose, “Investigation report on the method of pseudo random number generator system.” in Investigation Reports on Cryptographic Techniques, CRYPTREC, 2004
8. D. E. Knuth, ‘Deciphering a linear congruential encryption’, *IEEE Trans. Inf. Theory* **31** (1985), 49–52.

9. P. Lacharme, "Post-processing functions for a biased physical random number generator," in International Workshop on Fast Software Encryption, (Springer, 2008), pp. 334–342.
10. G. Marsaglia. "Diehard: A Battery of Tests of Randomness" (1996). http://www.stat.fsu.edu/pub/diehard/
11. NIST STS, "A Statistical Test Suite for the Validation of Random Number Generators and Pseudo-Random Number Generators for Cryptographic Applications" ,(2010) http://csrc.nist.gov/groups/ST/toolkit/rng/
12. Marianne Talbot. "A Romp Through Ethics for Complete Beginners (2012)". https://www.courses.com/university-of-oxford/a-romp-through-ethics-for-complete-beginners/1
13. M. Stipčević and Ç. K. Koç, "True Random Number Generators," in *Open Problems in Mathematics and Computational Science*, Ç. K. Koç, Ed. Cham: Springer International Publishing, 2014, pp. 275–315.
14. R. Boris and Z. Viacheslav, "The time-adaptive statistical testing for random number generators," in *2020 International Symposium on Information Theory and Its Applications (ISITA)*, Oct. 2020, pp. 344–347.
15. L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, S. D. Leigh, M. Levenson, M. Vangel, N. A. Heckert, and D. L. Banks, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," *NIST*, Sep. 2010.
16. A. A. Maksutov, P. N. Goryushkin, A. A. Gerasimov, and A. A. Orlov, "PRNG assessment tests based on neural networks," in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, Jan. 2018, pp. 339–341.
17. "Cracking Random Number Generators using Machine Learning – Part 1: Xorshift128," https://research.nccgroup.com/2021/10/15/cracking-random-number-generators-using-machine-learning-part-1-xorshift128/, Oct. 2021.
18. G. Amigo, L. Dong, and R. J. Marks Ii, "Forecasting Pseudo Random Numbers Using Deep Learning," in *2021 15th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE, 2021, pp. 1–7.
19. H. Kimura, T. Isobe, and T. Ohigashi, "Neural-Network-Based Pseudo-Random Number Generator Evaluation Tool for Stream Ciphers," in *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2019, pp. 333–338.
20. C. Li, J. Zhang, L. Sang, L. Gong, L. Wang, A. Wang, and Y. Wang, "Deep Learning-Based Security Verification for a Random Number Generator Using White Chaos," *Entropy*, vol. 22, no. 10, p. 1134, Oct. 2020.
21. J. L. Crespo, J. Gutierrez, and Á. Valle, "Random Number Generators quality assessment with Neural Networks," in *CASC 2023*, Havana, 2023.
22. J. L. Crespo, J. González-Villa, J. Gutierrez, and A. Valle, "Assessing the quality of Random Number Generators through Neural Networks," *Machine Learning: science and technology*. 5 (2024) 025072
23. S. Cong and Y. Zhou, "A review of convolutional neural network architectures and their optimizations," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 1905–1969, Mar. 2023.
24. Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022.
25. C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2023.
26. I. Nagy and A. Suciu, "Randomness Testing with Neural Networks," in *2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Oct. 2021, pp. 431–436.

27. E. B. Barker and J. M. Kelsey, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators,” National Institute of Standards and Technology, Tech. Rep. NIST SP 800-90Ar1, Jun. 2015.
28. M. Valle-Miñón, A. Quirce, A. Valle, and J. Gutiérrez, “Quantum random number generator based on polarization switching in gain-switched VCSELs,” *Optics Continuum*, vol. 1, no. 10, p. 2156, Oct. 2022.
29. A. Quirce and A. Valle, “Random polarization switching in gain-switched VCSELs for quantum random number generation,” *Optics Express*, vol. 30, no. 7, pp. 10 513–10 527, Mar. 2022.
30. https://github.com/fpgaminer/python-hmac-drbg (accedido el 21-8-2024) Library released under public domain.
31. D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science,” *Nature Communications*, vol. 9, no. 1, p. 2383, Jun. 2018.
32. D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” May 2016.
33. Riedmiller, M. and Braun, H., “A direct adaptive method for faster backpropagation learning: the RPROP algorithm” *IEEE International Conference on Neural Networks*, vol.1, pag 586-591 1993
34. I. Shparlinski, F. Voloch, “Generators of elliptic curves over finite fields”. Bulletin of the Institute of Mathematics Academia Sinica (New Series) Vol. 9 (2014), No. 4, pp. 657-670
35. Diederik P. Kingma and Jimmy Ba, “Adam: A Method for Stochastic Optimization”, 2017, https://arxiv.org/abs/1412.6980 (consultado el 21-8-24)