2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP) | 978-1-6654-0976-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCP53602.2021.9733481

Randomness Testing with Neural Networks

Imola Nagy
Technical University of
Cluj-Napoca,
Romania
Email: ngyimolanagy@gmail.com

Alin Suciu
Technical University of
Cluj-Napoca,
Romania
Email: Alin.Suciu@cs.utcluj.ro

***Abstract*—Testing the quality of data produced by Random and Pseudo-Random Number Generators (RNG-s and PRNG-s) is necessary for their safe use in cryptographic applications. Randomness is a probabilistic property and various statistical tests can be applied to evaluate the generated data sequences.**

This paper presents a novel approach for applying such tests, by training Artificial Neural Networks (ANN) to replicate the behaviour of both stand-alone and combined statistical tests. The process includes the development of various augmentation techniques used for creating a synthetic data set, the development and training of different ANN architectures and also the evaluation of the final classification models. The trained models can detect the presence or absence of diverse statistical characteristics in the generated data sequences.

The proposed solution reaches over 0.95 accuracy for both stand-alone and combined application of the following statistical tests from the NIST Statistical Test Suit [1]: Frequency Test, Frequency Test within a Block, Runs Test, Tests for the Longest-Run-of-Ones in a Block.

I. Introduction

Sequences of random numbers are essential elements of cryptography. Without them, encryption algorithms or secret key generation methods would become predictable and therefore insecure. Several tools have been developed for randomness testing [1][2][3][4], the current standard being the NIST-STS [1]. Different implementations exist for this test battery [5][6], which target the run time optimization of the original software.

The main objective of this paper is to investigate the ability of ANN-s to identify random binary sequences based on their statistical characteristics. ANN-s are suitable for such task because they can approximate both linear and nonlinear functions, thus being capable of modeling complex relationships. They can also estimate posterior probabilities, thus providing a basis for further statistical analysis[7]. However, ANN-s have their disadvantages. The accuracy of a model is highly dependent on the architecture, the hyper-parameters of the training, the quality and diversity of the data set. ANN-s also lack transparency in their decision-making, so they currently have limited use in the field of security.

II. Related work

Several recent attempts have been made to analyze randomness using ANN-s [8][9][10]. The presented approaches test the RNG-s and PNRG-s based on the ability of an ANN to approximate the underlying function of the generators and predict their future outputs. Moreover, the ANN's results were validated using standard statistical test suits [1][2] and the comparison confirms that ANN-s are powerful tools in the aspect of randomness analysis.

Similarly to the existing approaches, the proposed solution uses simple ANN architectures to tackle the identification of randomness. However, instead of predicting the future outputs of the RNG-s and PRNG-s, it focuses on creating classification models which can identify specific statistical features in the tested data sequences.

III. Conceptual Solution

The proposed solution aims to create classification ANN-s which can identify random sequences based on their statistical characteristics.

The classifiers have been created using *supervised learning* [11] for which it is necessary to generate and label a data set, to design the network architectures, to choose the training hyper-parameters, to train and evaluate the networks.

The *data* used for training the neural networks have been generated by a Quantum Random Number Generator (QRNG) [12]. The QRNG renders statistically random binary sequences, which have been validated by the NIST STS. In order to obtain a balanced data set, non-random sequences also have to be included. These have been derived from the random ones, using different *augmentation* algorithms [13] presented in Section III-A.

The ground-truth *labels* have been obtained by running NIST STS tests on the binary sequences and processing their outputs, detailed in Section III-B. In order to cover the different statistical characteristics, three types of *network architectures* have been designed, each representing a distinct approach for classification. They are presented in Section III-C The hyper-parameters of the network and the training process have been determined in the course of several experiments using different configurations, summarized in Section III-D.

A. Data augmentation techniques

In order to create an unbiased classifier, the data set has to be balanced, which means it has to contain an approximately equal number of samples from each class. In this case, the number of random and non-random sequences has to be the same.

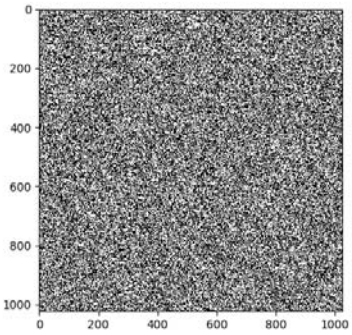


Fig. 1. Visualization of binary sequences produced by the QRNG.

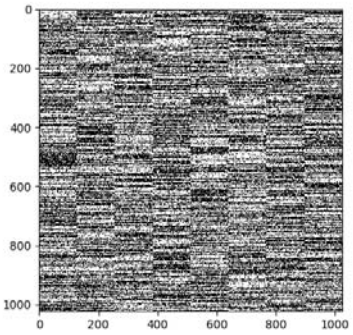
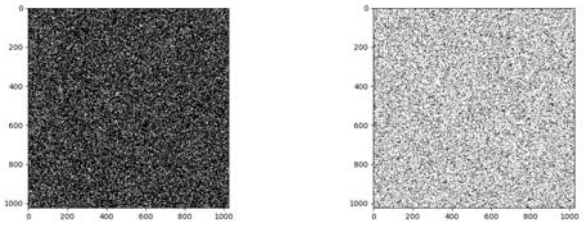


Fig. 3. Visualization of augmented binary sequences after changing the frequency of bits within blocks in them. Modified bit ratio = 60%



(a) Sequences unbalanced toward 0 (b) Sequences unbalanced toward 1

Fig. 2. Visualization of augmented binary sequences after changing the frequency of bits in them. Modified bit ratio = 50%

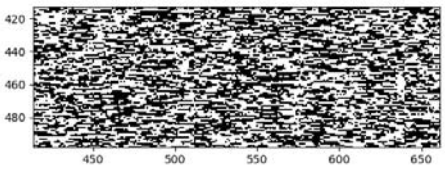


Fig. 4. Visualization of augmented binary sequences after altering the length of runs in them. Modified bit ratio = 40%

The non-random sequences have been derived from the true-random data produced by the QRNG (Figure 1), by applying different augmentation techniques. The algorithms increase the data set by appending modified copies of existing samples to it, aiming to disturb a specific statistical property of the entry sequences. The results of the augmentation techniques are visualized by Figures 2 - 7. The plots represent snippets of vertically stacked bit sequences after the augmentation. The bits of 0 are colored black and the bits of 1 white.

The augmentation algorithms are the following:

1) Changing the frequency of bits: Under the assumption of randomness, the number of zeros and ones has to be approximately the same in a binary sequence. The purpose of this algorithm is to disrupt this balance. It does so by setting a fraction of the sequence's bits to 0, if the sequence has to be unbalanced toward 0 and to to 1 otherwise.

Figure 2 illustrates the effects of the augmentation algorithm on the true-random data.

2) Changing the frequency of bits in blocks: In a random sequence the number of zeros and ones has to be approximately the same in its smaller sub-sequences. The algorithm divides the input sequence into blocks of M bits, and aims to disrupt the balance between the number of 0 and 1 bits, without changing their balance in the whole sequence.

50% of randomly chosen blocks are unbalanced toward 0 and the other 50% towards 1, using the method described in III-A1. Figure 3 illustrates the effects of the augmentation algorithm on the true-random data.

3) Altering the length of runs: A run is an uninterrupted sequence of identical bits, bounded both at the beginning and at the end by a bit of the opposite value. A true-random sequence has many small runs and the purpose of this augmentation technique is to create fewer and larger ones.

The algorithm divides the input sequence into blocks of M bit, and splits them further into two sub-blocks, from which one gets unbalanced toward 0 and the other toward 0, by using the method described in III-A1. At the end, the runs are scrambled within each block to remove the otherwise present chequerboard pattern.

Figure 4 illustrates the effects of the augmentation algorithm on the true-random data.

4) Increasing the length of the longest run from each block: The purpose of this augmentation technique is to increase the size of the longest run within each block, without changing the balance of the bits within the block or in the whole sequence.

The algorithm divides the input sequence into blocks of M bits and inserts in each of them a run of zeros and a run of ones, which have configurable lengths. Figure 5 illustrates the effects of the augmentation algorithm on the true-random data.

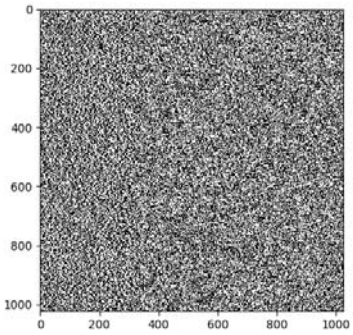


Fig. 5. Visualization of augmented binary sequences after increasing the length of the longest run from each block in them.

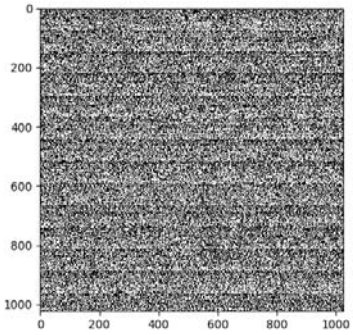


Fig. 6. Visualization of binary sequences after augmentation with 9 bit templates.

5) Augmentation with aperiodic templates: The purpose of the algorithm is to simulate a generator producing repetitive binary sequences. It fills in the input sequence with patterns from the NIST STS's aperiodic template set [1]. The set is grouped by a length of the templates, which vary between 2 and 16 bits.

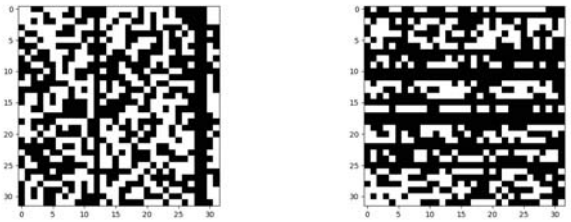
The input sequences are decorated with one of the groups, by inserting a configurable number of aperiodic templates into them. Figure 6 illustrates the effects of the augmentation algorithm on the true-random data.

6) Changing the ranks of binary matrices: The algorithm transforms the input sequence into an array of binary matrices with size M^2. Under the assumption of randomness, the majority of the matrices should have the ranks of M and M-1.

The purpose of this augmentation technique is to increase the number of matrices having ranks less than M-1. This is done by setting to 0 a configurable number of rows (Figure 7a) and columns (Figure 7b) in *mbr %* of the sequence's matrices.

B. Labeling the data set

The ground truth labels of the data set have been determined by running statistical tests from the NIST STS [1] using its optimized implementation [5] and evaluating the resulting P-values [14].



(a) Matrix with zeroed columns (b) Matrix with zeroed rows

Fig. 7. Visualization of matrices from an augmented binary sequence after reducing their rank.

The statistical tests used for labeling are: *Frequency (Monobit) Test*, *Frequency Test within a Block*, *Runs Test*, *Tests for the Longest-Run-of-Ones in a Block*, *The Binary Matrix Rank Test*, *The Discrete Fourier Transform (Spectral) Test*, *The Non-overlapping Template Matching Test* and *The Cumulative Sums Test*.

Each test returns one or more P-values for each input binary sequence. The P-value represents the probability that a true-random generator would have yielded a less-random data sequence, with regard to the tested statistical features[1]. It can have values from the [0, 1] interval, 1 meaning the input sequence is true-random and 0 that is not random at all. The values between the two extremities are categorised using an α threshold. For P-value $\geq \alpha$, the input sequence is considered 'random enough', otherwise non-random. α is usually chosen from the [0.001, 0.01] interval.

The ground truth labels have been determined by setting $\alpha = 0.01$, if P-value $\geq \alpha$ then label = 1, otherwise label = 0. For the networks detecting more than one statistical feature, the labels have been concatenated into a single array. Eg. if the sequences are labeled for the Frequency and Block Frequency Tests, the label length is 2, where the first position indicates if the sequence passes the Frequency Test and the second position if it passes the Block Frequency one.

C. Architectures

1) Fully Connected Architecture: The Fully Connected Neural Network (FCNN) or multi-layer perceptron (MLP) is a type of architecture which is composed solely of Fully connected (Dense) layers. It must have at least three layers: an input, an output and a hidden layer.

In the proposed architecture the number of hidden layers, the size of the input, output and first hidden layer are configurable. The dimensions of the other hidden layers' are half of the one to which they are connected. The hidden layers have ReLU activation functions, while the output layer has Sigmoid.

2) Convolutional Architecture: Convolutional Neural Networks (CNNs) aim to condense input data into an easier-to-process form without losing features that are essential to

having a good prediction. CNNs generally have two parts: a feature extractor and a classifier.

In the proposed architecture, the feature extractor has 10 convolutional blocks, each block being made of a 1D Conv layer, a ReLU activation function and a 1D MaxPooling layer. All Conv layers have a kernel size of 2^l x 3, where the l is the index of the convolution block. The kernel dimension is chosen in accordance with the template size of the Non-Overlapping Template Matching Test for which the architecture has been applied. All Pooling layers reduce the width of the feature map by a factor of 2. These operation make the features grow on the channel dimension and shrink on the width dimension.

The classifier part has a Fully Connected layer with Sigmoid activation.

3) LSTM Architecture: The Long-Short Term Memory architecture is a Recurrent Neural Network (RNN), designed to learn relevant features from long input data, processed in sequences [15].

The proposed architecture has two parts: a recurrent section and a classifier. The recurrent section consists of LSTM layers with ReLU activation functions, while the classification section is made of a Fully Connected layer and a Sigmoid activation. The number of LSTM layers, the size of the input and output layer, as well as the size of the first LSTM layer are configurable. The dimensions of the other LSTM layers' are half of the one to which they are connected.

D. Training parameters

In order to train the architectures, the labeled data set has been divided into training (60%), validation (20%) and testing (20%) sets. The training set is used for updating the weights during training, the validation for fine-tuning the network's hyper-parameters and the testing set is used for the final evaluation of the model. The training batch sizes vary from 64 to 256 sequences and the testing batch size is set to 32.

The chosen loss function is the *Mean Squared Error* (MSE) for all experiments and the optimizer is the *adaptive moment estimation* (Adam) [16].

Each model is trained until the loss calculated on the train and validation sets starts to converge or when it becomes clear from the learning curves that the model has stopped learning new features (the loss stagnates).

IV. Experimental Evaluation and Results

Table I presents the data sets created to identify the different statistical characteristics in the binary sequences. Each entry defines the raw input data size, the sequence length, the necessary augmentation techniques and the final label size of the set. The label size also defines the tackled classification problem: when the label size is 1, the network performs binary classification and for label size ≥ 1, multi-class classification. The name indicates the NIST STS tests corresponding to the targeted statistical characteristic. The combination of tests is composed of the Frequency, Block Frequency, Runs and Longest-Run-of-Ones Tests, their corresponding labels appear in this order in the output.

Table II summarises the most successful experiments performed for each test. The created networks have been trained on the training and validation data sets and evaluated by their accuracy and run time on the test set.

The most suitable architecture types and sizes have been identified using an incremental approach, by testing different layer alignments and analysing the produced *learning curves* as well as the final outputs. The learning curves represent the values of the loss function calculated on the train - and validation set in each epoch. They reflect both the quality of the data set and the suitability of the architecture for the learning task. If they both start to converge towards a global minimum, means that the network learned all the features it could and the learning can be stopped. Figure 9 shows an example of the learning curves produced by Fully Connected architectures having one, two and three hidden layers.

The best results have been achieved by the network experiments labeled by the *Frequency (Monobit) Test*, *Block Frequency Test*, *Runs Test*, *Tests for the Longest-Run-of-Ones in a Block*, using the Fully Connected architecture, producing an accuracy over 0.95. They were also fused into a single Combined network, which can simultaneously test the four different statistical features. It obtains a final accuracy of 0.96 and has the same run time as a single-test network.

The experiments targeting to learn the features labeled by the *Discrete Fourier Transform (Spectral) Test* and the *Cumulative Sums Test* reach over 0.84 accuracy with the Fully Connected architectures. Although this is a promising result considering the complexity of the test, the final classifiers are not precise enough for practical applications.

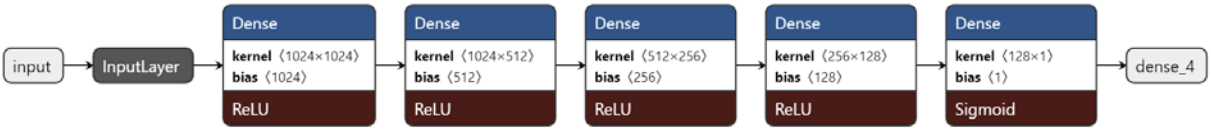
The LSTM architecture has been used for the *Binary Matrix Rank Test* in order to tackle the problem of the long input sequences, but it does not produce acceptable results, the final accuracy remaining poor.

The experiments targeting to reproduce the *Non-overlapping Template Matching Test* are using convolutional architectures, as they are suitable for pattern recognition tasks. During the training process, it has been observed that the networks are learning way too fast. This indicates that the proposed data set might be biased and the classifier favours the majority class, not being able to learn the correct features.

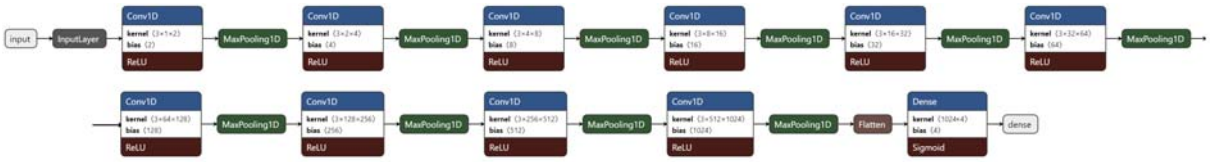
V. Conclusion

The proposed solution analyses the ability of neural networks to recognize random binary sequences based on their statistical characteristics. Due to the fact that the developed classification models only approximate the statistical tests of the NIST STS, they achieve good performance, but lower accuracy compared to the classical approaches. As a future development, sturdier architecture can be made to tackle more complex tests.

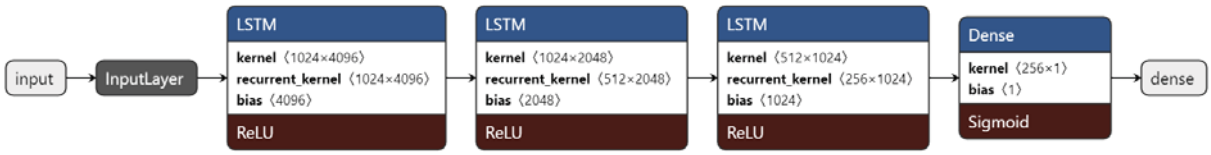
Since the labeling of the data set has been done by applying the tests of the NIST STS, the trained networks cannot overturn the classical approach. However, analysing the sequences with ANN-s can help to discover new correlations between the tests and aid in the construction of new ones.



(a) Fully Connected Architecture with 3 hidden layers



(b) Convolutional Architecture with 10 Conv layers



(c) LSTM Architecture with 10 LSTM layers

Fig. 8. Architecture diagrams

TABLE I
DATA SET STRUCTURE FOR DIFFERENT EXPERIMENTS

SP = Sequence Percentage - represents the percentage of sequences from the input data on which the augmentation algorithm is applied; *mbr = modified bit ratio* - denotes the percentage of bits affected by the algorithm; *M = block size*; *m = template size*; *min/max-ins-run/temp = minimum and maximum length of the inserted runs/templates*; *max-ins-rc = maximum number of inserted rows/columns*.

Test name	Raw data size	Sequence length (bits)	Augmentation Applied	Label Size
Frequency	1MB	1024	Change the bit frequency; SP=85%; mbr = 50%	1
Block Frequency	1MB	1024	Change the bit frequency in blocks; SP=85%; M = 128; mbr = 60%	1
Runs	1MB	1024	Altering the length of runs; SP=85%; M = 128; mbr = 40%	1
Longest Run of Ones	1MB	1024	Increasing the length of the longest runs; M = 128; min-ins-run = 15; max-ins-run = 35	1
Ranks	10MB	38912	Changing the ranks of binary matrices; SP=100%; M = 32; mbr = 50%; max-ins-rc = 10	1
Spectral	1MB	1024	Change the bit frequency; SP=35%; mbr = 50% Change the bit frequency in blocks; SP=30%; M = 128; mbr = 60% Altering the length of runs; SP=85%; M = 128; mbr = 35%	1
Non-Overlapping Aperiodic Templates, m=3	1MB	1024	Augmentation with aperiodic templates; SP=100%; m = 3; min-ins-temp=25; max-ins-temp=150	4
Non-Overlapping Aperiodic Templates, m=9	1MB	1024	Augmentation with aperiodic templates; SP=100%; m = 9; min-ins-temp=25; max-ins-temp=150	148
Cumulative Sums	1MB	1024	Change the bit frequency; SP=35%; mbr = 50% Change the bit frequency in blocks; SP=45%; M = 128; mbr = 60% Altering the length of runs; SP=45%; M = 128; mbr = 35%	2
Combination of tests	10MB	1024	Change the bit frequency; SP=85%; mbr = 50% Change the bit frequency in blocks; SP=85%; M = 128; mbr = 60% Altering the length of runs; SP=85%; M = 128; mbr = 35% Increasing the length of the longest runs; M = 128; min-ins-run = 15; max-ins-run = 35	4

Different augmentation algorithms have been designed and implemented in order to generate synthetic data sets for training. They aim to disturb specific statistical characteristics in the true-random binary sequences in order to make sure that the classification models are learning based on the correct features. The augmentation can be further fine-tuned to obtain a data set with better coverage.

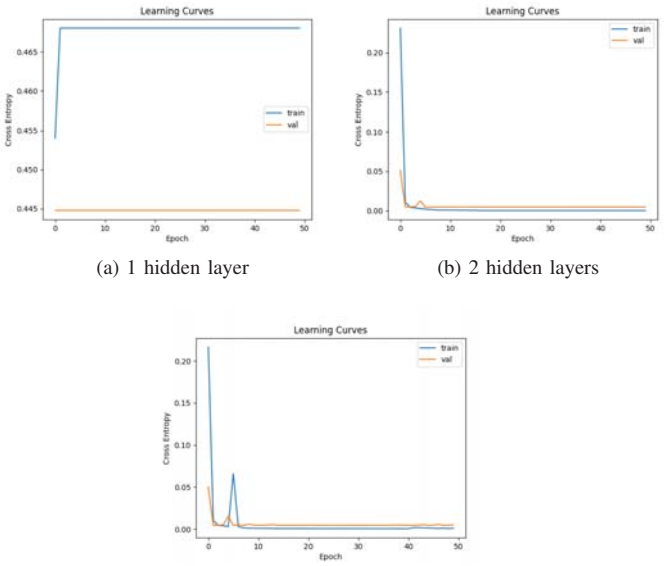
REFERENCES

[1] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, S. Leigh, M. Levenson, M. Vangel, N. Heckert, and D. Banks, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," 2010-09-16 2010. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762

[2] G. Marsaglia, "The marsaglia random number cdrom including the diehard battery of tests of randomness," 1995. [Online]. Available: https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard

TABLE II
BEST RESULTS FOR EACH STATISTICAL CHARACTERISTIC EXPERIMENT

Test name	Architecture	Hidden layers	Input size (bits)	First Hidden Size	Output (Label) size	Training Batch size	Loss	Accuracy	Run time/ batch (batch=32)
Frequency	Fully Connected	2	1024	512	1	256	0.0061	0.99	2 ms
Block Frequency	Fully Connected	3	1024	512	1	256	0.0047	0.99	2 ms
Runs	Fully Connected	1	1024	512	1	256	0.0330	0.96	2 ms
Longest Run of Ones	Fully Connected	4	1024	512	1	256	0.0134	0.98	4 ms
Ranks	LSTM	1	(38, 1024)	1024	1	64	0.4664	0.53	0.81 s
Spectral	Fully Connected	2	1024	512	1	256	0.1357	0.84	2 ms
Non-Overlapping Aperiodic Templates m=3	Convolutional	10	1024	512	4	256	0.2191	0.65	0.98 s
Non-Overlapping Aperiodic Templates m=9	Convolutional	10	1024	512	148	256	0.0390	0.96	0.98 s
Cumulative Sums	Fully Connected	3	1024	512	2	256	0.0240	0.87	2 ms
Combination of tests	Fully Connected	4	1024	256	4	256	0.0112	0.96	2 ms



(a) 1 hidden layer

(b) 2 hidden layers

(c) 3 hidden layers

Fig. 9. Learning curves of the Block Frequency experiments, using Fully Connected architectures.

[3] R. G. Brown, "Dieharder: A random number test suite," 2021. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[4] P. L'Ecuyer and R. Simard, "Testu01: A c library for empirical testing of random number generators," *ACM Trans. Math. Softw.*, vol. 33, no. 4, Aug. 2007. [Online]. Available: https://doi.org/10.1145/1268776.1268777

[5] M. Sýs and Z. Riha, "Faster randomness testing with the nist statistical test suite," vol. 8804, 10 2014.

[6] M. Sýs, Z. Říha, and V. Matyáš, "Algorithm 970: Optimizing the nist statistical test suite and the berlekamp-massey algorithm," *ACM Trans. Math. Softw.*, vol. 43, no. 3, Dec. 2016. [Online]. Available: https://doi.org/10.1145/2988228

[7] G. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[8] F. Fan and G. Wang, "Learning from pseudo-randomness with an artificial neural network–does god play pseudo-dice?" *IEEE Access*, vol. 6, pp. 22 987–22 992, 2018.

[9] Y. Feng and L. Hao, "Testing randomness using artificial neural network," *IEEE Access*, vol. 8, pp. 163 685–163 693, 2020.

[10] 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, 2020. [Online]. Available: https://www.mdpi.com/1099-4300/22/10/1134

[11] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 1995, vol. 2.

[12] M. M. Jacak, P. Jóźwiak, J. Niemczuk, and J. E. Jacak, "Quantum generators of random numbers," *Scientific Reports*, vol. 11, 2021.

[13] D. A. van Dyk and X.-L. Meng, "The art of data augmentation," *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001. [Online]. Available: https://doi.org/10.1198/10618600152418584

[14] M. Sýs, Z. Riha, V. Matyas, K. Marton, and A. Suciu, "On the interpretation of results from the nist statistical test suite," *Romanian Journal of Information Science and Technology*, vol. 18, no. 1, p. 18–32, 2015.

[15] L. C. Jain and L. R. Medsker, *Recurrent Neural Networks: Design and Applications*, 1st ed. USA: CRC Press, Inc., 1999.

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.