Hindawi
Security and Communication Networks
Volume 2020, Article ID 4241713, 18 pages
https://doi.org/10.1155/2020/4241713

Research Article

High-Efficiency Min-Entropy Estimation Based on Neural Network for Random Number Generators

Na Lv,[1,2,3] **Tianyu Chen**,[1,2] **Shuangyi Zhu**,[1,2,3] **Jing Yang**,[4] **Yuan Ma**,[1,2] **Jiwu Jing**,[5] **and Jingqiang Lin**[1,2]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*
[2]*Data Assurance and Communications Security Research Center, Chinese Academy of Sciences, Beijing 100093, China*
[3]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100093, China*
[4]*School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100093, China*
[5]*China Information Technology Security Evaluation Center, Beijing 100085, China*

Correspondence should be addressed to Tianyu Chen; chentianyu@iie.ac.cn

Received 30 November 2019; Accepted 22 January 2020; Published 17 February 2020

Academic Editor: Clemente Galdi

Random number generator (RNG) is a fundamental and important cryptographic element, which has made an outstanding contribution to guaranteeing the network and communication security of cryptographic applications in the Internet age. In reality, if the random number used cannot provide sufficient randomness (unpredictability) as expected, these cryptographic applications are vulnerable to security threats and cause system crashes. Min-entropy is one of the approaches that are usually employed to quantify the unpredictability. The NIST Special Publication 800-90B adopts the concept of min-entropy in the design of its statistical entropy estimation methods, and the predictive model-based estimators added in the second draft of this standard effectively improve the overall capability of the test suite. However, these predictors have problems on limited application scope and high computational complexity, e.g., they have shortfalls in evaluating random numbers with long dependence and multivariate due to the huge time complexity (i.e., high-order polynomial time complexity). Fortunately, there has been increasing attention to using neural networks to model and forecast time series, and random numbers are also a type of time series. In our work, we propose several new and efficient approaches for min-entropy estimation by using neural network technologies and design a novel execution strategy for the proposed entropy estimation to make it applicable to the validation of both stationary and nonstationary sources. Compared with the 90B's predictors officially published in 2018, the experimental results on various simulated and real-world data sources demonstrate that our predictors have a better performance on the accuracy, scope of applicability, and execution efficiency. The average execution efficiency of our predictors can be up to 10 times higher than that of the 90B's for 10^6 sample size with different sample spaces. Furthermore, when the sample space is over 2^2 and the sample size is over 10^8, the 90B's predictors cannot give estimated results. Instead, our predictors can still provide accurate results. Copyright© 2019 John Wiley & Sons, Ltd.

1. Introduction

Random number generator (RNG) is a fundamental and important element in modern cryptography, which especially provides a basic guarantee for the security of the network and communication system, as in [1–4]. The output of RNGs, called random number, is widely used in a large number of security and cryptographic applications. These applications include the generation of cryptographic keys, initialization vectors in cryptographic algorithm, digital signature generation, and nonces and padding values. If the output of RNGs cannot provide sufficient unpredictability as expected, the cryptographic applications would be vulnerable, as in [5–7]. Thus, the necessity of the security analysis on RNGs is self-evident, especially it is important to evaluate the quality of the entropy source which is the main source of randomness for RNGs.

At present, in order to guide the designers, users, and assessors to analyze the security of RNGs, many research organizations or individuals have provided a number of approaches for testing and evaluating the RNGs. These approaches can be roughly divided into two classes: statistical property test and entropy estimation. Specifically, the statistical property test is proposed at first, such as the NIST Special Publication 800-22 [8], AIS 31 [9], Diehard battery [10], and TestU01 [11], which detects whether the output sequence has obvious statistical defects. Because it only focuses on the statistical properties of outputs rather than the internal structure and generation principle of RNGs, the statistical property test is a universal (black-box) testing method for various types of generators, and it is easy to operate. With in-depth understanding for the randomness in the past few years, the concept of "entropy" has been proposed to evaluate the security of RNGs. Entropy is a measure of uncertainty that is appropriate to reflect the amount of randomness. Therefore, several major standardization organizations' criterions recommend to adopt the entropy to quantify the randomness (unpredictability) of the outputs of RNGs, such as the ISO/IEC 18031 [12] and AIS 31 [9]. There are many types of methods for measuring entropy, including Shannon entropy, Rényi entropy, and min-entropy. Min-entropy is a very conservative measure, which means the difficulty of guessing the most-likely output of entropy sources [13].

However, the entropy estimation of entropy sources is a very challenging task because the common assumptions may not be consistent with the real conditions and the distribution of the outputs is unknown. Nowadays, there are two ways to implement entropy estimation: theoretical entropy estimation (stochastic model) and statistical entropy estimation. A theoretical proof for the security of RNGs can be achieved from a suitable stochastic model, as in [14–17]. But the modeling is always difficult and complex, because it is always based on the specific structure of a RNG and an appropriate assumption on the entropy source's behavior, and even some structures of RNGs still do not have a suitable model [18–21]. Relatively, statistical entropy estimation is still based on the idea of entropy estimation, but it is implemented by means of statistical black-box testing, which has a good applicability for evaluating various types of RNGs. Thus, statistical entropy estimation can partly solve the problem that the entropy of some RNGs cannot be quantified by modeling.

The NIST Special Publication 800-90B [22] (called 90B in the text below) is a typical representative of the statistical entropy estimations, which is based on min-entropy and specifies how to design and test entropy sources. The final version of the 90B was officially published in January 2018 [23] and replaces the second draft of 90B published in 2016 [22]. The predictors proposed by Kelsey et al. [24] have a better performance than the other estimators in this standard and refer to a machine learning algorithm that attempts to predict each sample in a sequence and updates its internal state based on all observed samples. However, there are some problems in these 90B's predictors. On the one hand, every predictor is designed to perform well only for the sources with a certain statistical property as stated in [24], which constrains the application scope of the predictors. On the other hand, the execution efficiency of these predictors is influenced significantly when the selected sample space is large. It is analyzed that the time complexity of 90B's predictors has high order linear relationship with the size of sample space. In the released C++ code of 90B's estimators published in 2018, the bits per symbol of the samples are still limited to between 1 and 8, inclusive, in order to prevent too low execution efficiency. Therefore, they are not likely to be well applied to the entropy evaluation of entropy sources with unknown statistical behaviors, multivariate, and long-range correlation.

As we know, the output sequences of RNGs are also a type of time series. Fortunately, there has been increasing attention to using neural networks to model and forecast time series, which has been found to be an alternative method when compared with various traditional time series models [25–27]. Some specific neural networks are applicable to the prediction of time series via approximating the probability distribution function (PDF), and the time complexity varies linearly with the sample space. Feedforward neural networks (FNNs) and recurrent neural networks (RNNs) are the typical representatives. FNNs are the quintessential deep learning models. In 1991, de Groot and Würtz [28] presented a detailed analysis of univariate time series forecasting using FNNs for two nonlinear time series. FNNs are used to approximate some PDFs [29]. RNNs are a family of neural networks for processing sequential data, which can also be used for time series forecasting [30–32]. Therefore, it is worthwhile and feasible to study new methods of entropy estimation for RNG based on neural networks.

1.1. Motivation. In this paper, we aim to propose several suitable and efficient predictors for the security evaluation of RNGs, especially for the min-entropy estimation of sequences generated by entropy sources. Since both FNN and RNN are suitable for time series prediction, we design two predictors based on these neural network models. The advantages of the proposed predictors are roughly described as follows. Because the selected neural network models have good universality for the prediction of various types of time series in principle, the designed predictors have wide applicability for the entropy estimation of the output of entropy sources. Moreover, neural network-based predictors have high execution efficiency, which can properly handle the difficulty in evaluating random numbers with long dependence and multivariate due to the huge time complexity.

1.2. Contributions. In summary, we make the following contributions:

(i) We are the first to adopt neural network models to design predictors to estimate min-entropy for RNG and propose a suitable execution strategy which makes our approach applicable to predict both

stationary and nonstationary sequences generated by different entropy sources.

(ii) We conduct a series of experiments to verify the accuracy of our predictors by using many typical simulated sources where the theoretical entropy can be obtained from the known probability distribution. Additionally, the computational complexity are evaluated theoretically. The results show that our approaches enable the entropy to be estimated with an error of less than 6.55%, and the error is up to 14.65% for 90B's predictors. The time complexity of our estimation is a linear relationship with sample space, which is high-order linear relationship with sample space for the 90B.

(iii) We experimentally compare the advantages of our predictors over 90B's predictors on accuracy, application scope, and execution efficiency. The experimental datasets include several typical real-world data sources and various simulated datasets. The experimental results indicate our predictors have higher accuracy, execution efficiency, and wider scope of applicability than those of 90B's predictors. Furthermore, when the test sample space and sample size are continuously growing, the execution efficiency of 90B's predictors becomes too low to estimate the entropy within an acceptable time interval, while our proposed predictors can still calculate the estimated results efficiently.

The rest of the paper is organized as follows. In Section 2, we introduce fundamental definitions about min-entropy, the evolution of the 90B and estimators especially predictors defined in this criterion, and two typical neural networks we choose to support our research. In Section 3, we propose two predictors based on neural networks for min-entropy estimation, design an execution strategy, and give the accuracy verification and complexity analysis of our predictors. Furthermore, we apply our predictors to different types of simulated and real-world data sources and compare the advantages of our predictors over 90B's predictors on accuracy, application scope, and execution efficiency in Section 4. We finally conclude our work in Section 5.

2. Preliminaries

In this section, firstly, we introduce the fundamental concept of min-entropy, which is the core mathematical thought and assessment method in our work. Then, we introduce the evolution process of the 90B and relevant research work on this criterion. After that, we introduce the estimators defined in the 90B, especially the predictive model-based estimators, which are the focus of this paper. At last, we describe two predictive models based on neural networks, which apply to time series forecasting and contribute to the design of new predictors in our work.

2.1. Min-Entropy of Entropy Source. The concept of entropy is the core mathematical thought of the 90B and min-entropy is the assessment method, which is a conservative way to ensure the quality of random numbers in the worst case for some high-security applications, such as the seed of PRNGs. The 90B [22] gives the definition of min-entropy: the min-entropy of an independent discrete random variable X that takes values from the set $A = \{x_1, x_2, \ldots, x_k\}$ ($k \in \mathbb{Z}^*$ denotes the size of sample space), with the probability $\Pr(X = x_i) = p_i (i = 1, 2, \ldots, k)$. The min-entropy of the output is

$$H_{\min} = \min_{1 \le i \le k} (-\log_2 p_i) = -\log_2 \left(\max_{1 \le i \le k} (p_i) \right). \tag{1}$$

If X has min-entropy H, then the probability of observing any particular value for X is no greater than 2^{-H}. The maximum possible value for the min-entropy of a random variable with k distinct values is $\log_2(k)$, which is attained when the random variable has a uniform probability distribution, namely, $p_1 = p_2 = \cdots = p_k = 1/k$.

For the non-IID source, such as Markov process, Turan et al. provided a calculation method of min-entropy in [22]. A stochastic process $\{X_i\}_{i \in \mathbb{N}}$ that takes values from the finite set A defined above is known as a first-order Markov chain, if

$$\Pr\left(X_{m+1} = x_{m+1} \mid X_m = x_m, X_{m-1} = x_{m-1}, \ldots, X_0 = x_0\right) = \Pr\left(X_{m+1} = x_{m+1} \mid X_m = x_m\right), \tag{2}$$

for any $m \in \mathbb{Z}^*$ and all $x_0, x_1, \ldots, x_m, x_{m+1} \in A$. In a d^{th}-order Markov process, the transition probabilities have the property that

$$\Pr\left(X_{m+1} = x_{m+1} \mid X_m = x_m, X_{m-1} = x_{m-1}, \ldots, X_0 = x_0\right) = \Pr\left(X_{m+1} = x_{m+1} \mid X_m = x_m, \ldots, X_{m-d+1} = x_{m-d+1}\right). \tag{3}$$

The initial probabilities of the process are $p_i = \Pr(X_0 = i)$, and the transition probabilities are $p_{ij} = \Pr(X_{m+1} = j \mid X_m = i)$. The min-entropy of a Markov process of length L is defined as

$$H_{\min} = -\log_2 \left(\max_{x_1, \ldots, x_L} p_{x_1} \prod_{j=1}^{L} p_{x_{j-1} x_j} \right). \tag{4}$$

The approximate value of min-entropy per sample can be obtained by dividing $H_{\min}$ by L.

2.2. NIST SP 800-90B and Its Entropy Estimation. The 90B is a typical case that evaluates the quality of the entropy source from the perspective of min-entropy. The evolution process of 90B mainly includes the following three stages. Compared with the second draft, the final version in January 2018 has made some corrections.

(i) The first draft of 90B [13] was published in August 2012, which included five estimators proposed by Hagerty and Draper [33]. These estimators are

collision test, partial collection test, Markov test, compression test, and frequency test, which are suitable for sources that do not necessarily satisfy the IID assumption. But these estimators give significant underestimates which were found by Kelsey et al. [24] through experiments.

(ii) Subsequently, the 90B was updated to the second draft [22] in January 2016, and the estimators based on predictors which were proposed by Kelsey et al. for the first time were adopted. Compared with the first draft, the second draft has the following main changes. (1) Among these estimators, the partial collection test in the first draft was deleted, and the frequency test in the first draft was replaced by most common value estimator, and two new estimators including t-tuple estimator and longest repeated substring estimator were added. (2) The second important update was the addition of four predictors for entropy estimation. However, the underestimation problem was not solved in the second draft. Zhu et al. [34] proved the underestimation problem for non-IID data from theoretical analysis and experimental validations and proposed an improved method.

(iii) The final official version of 90B [23] was published in January 2018. In the final 90B, estimators with significant underestimates, such as collision estimator and compression estimator, are modified to be limited to only for binary inputs, which may reduce the overall execution efficiency of min-entropy estimation for nonbinary inputs. In addition, the calculation process and method of key variables for min-entropy estimation are also corrected, such as P'_{global}, min-entropy of each predictor.

2.2.1. Execution Strategy of Min-Entropy Estimation in 90B. The 90B takes the following strategy to estimate the min-entropy. It first checks whether the tested datasets are IID or not. On the one hand, if the tested datasets are non-IID, there are ten estimators as mentioned in Section 2.2.2 for entropy estimation. Each estimator calculates its own estimation independently, then among all estimations, the minimum one is selected as the final estimation result for the entropy source. On the other hand, if the tested datasets are considered IID, only the most common value estimator is employed. Finally, it applies restart tests and gives the entropy estimation.

Note that, this article only focuses on the analysis and comparison with the 90B's four predictors, and the research on other parts of the 90B is not considered in this study.

2.2.2. Estimators in 90B. In the final NIST SP 800-90B, there are ten estimators, and each estimator has its own specific characteristics. According to the underlying methods they employed, we divide these estimators into three classes: frequency-based type, entropy statistic based type, and predictors based type. The following is a brief introduction of the ten estimators, and the details can be found in [23].

(1) Predictor-Based Type Estimators. The followings are four predictors proposed by Kelsey et al. [24] for entropy estimation for the first time. Kelsey et al. utilized several machine learning models served as predictors to improve the accuracy of entropy estimation. But these predictors perform well only for specific distributions:

(i) *Multi Most Common in Window (MultiMCW) Predictor*. This predictor performs well in cases where there is a clear most common value, but that value varies over time.

(ii) *Lag Predictor*. The Lag subpredictor predicts the value that occurred N samples back in the sequence. This predictor performs well on sources with strong periodic behavior, if N is close to the period.

(iii) *Multi Markov Model with Counting (MultiMMC) Predictor*. The MultiMMC subpredictor predicts the most common value followed the previous N sample string. The range of the parameter N is set from 1 to 16. This predictor performs well on data from any process that can be accurately modeled by an N^{th}-order Markov model.

(iv) *LZ78Y Predictor*. This predictor performs well on the sort of data that would be efficiently compressed by LZ78-like compression algorithms.

(2) Other Estimators. Four frequency-based type estimators and two entropy statistic-based type estimators are described as follows. The min-entropy of the former type of estimators is calculated according to the probability of the most-likely output value, and the other is based on entropic statistics presented by Hagerty and Draper [33]. Among them, three estimators, including Markov estimator, Collision estimator, and Compression estimator, explicitly state that they only apply to binary inputs in the final 90B published in 2018, which may reduce the execution efficiency for nonbinary inputs as proved through experiments in Section 4.3.

(i) *Most Common Value Estimate*. This estimator calculates entropy based on the number of occurrences of the most common value in the input dataset and then constructs a confidence interval for this proportion. The upper bound of the confidence interval is used to estimate the min-entropy per sample of the source.

(ii) *Markov Estimate*. This estimator computes entropy by modeling the noise source outputs as a first-order Markov model. The Markov estimate provides a min-entropy estimate by measuring the dependencies between consecutive values from the input dataset. This method is only applied to binary inputs.

(iii) *T-Tuple Estimate*. This method examines the frequency of *t-tuples* (pairs, triples, *etc.*) that appears in the input dataset and produces an estimate of the entropy per sample, based on the frequency of those *t-tuples*.

(iv) *Longest Repeated Substring Estimate (LRS estimate).* This method estimates the collision entropy (sampling without replacement) of the source, based on the number of repeated substrings (tuples) within the input dataset.

(v) *Collision Estimate.* This estimator calculates entropy based on the mean number of samples to see the first collision in a dataset, where a collision is any repeated value. This method is only applied to binary inputs.

vi) *Compression estimate.* This estimator calculates entropy based on how much the tested data can be compressed. This method is also only applied to binary inputs.

2.2.3. Min-Entropy Estimation of 90B's Predictors. Each predictor in the 90B attempts to predict the next sample in a sequence according to a certain statistical property of previous samples and provides an estimated result based on the probability of successful prediction. Every predictor consists of a set of subpredictors and chooses the subpredictor with the highest rate of successful predictions to predict the subsequent output. As for each predictor, it calculates the global predictability and local predictability with the upper bound of the 99% confidence interval and then derives the global and the local entropy estimations, respectively. Finally, the final entropy estimation for this predictor is the minimum of the global and the local entropy estimations.

For estimating the entropy of a given entropy source, each predictor offers a predicted result after testing the outputs produced by the source and provides an entropy estimation based on the probability of successful predictions. After obtaining the estimations from the predictors, the minimum estimation of all the predictors is taken as the final entropy estimation of the entropy source.

The entropy estimation will be too loose, if there is no predictor applied to detect the predictable behaviors. But if a set of predictors with different approaches are applied, they can guarantee that the predictor which is the most effective at predicting the entropy source's outputs determines the entropy estimation.

2.3. Two Predictive Models Based on Neural Networks. Next, we will introduce two main neural network models to help us design predictors for entropy estimation, feedforward neural networks (FNNs), and recurrent neural networks (RNNs), respectively.

2.3.1. FNNs. The goal of a feedforward network is to approximate some function f^*. For an instance, a classifier $Y = f^*(X)$ maps an input X to a category Y. A feedforward network describes a mapping $Y = f(X;\theta)$ and learns the value of the parameters θ that result in the best function approximation. The principle of FNN is depicted in Figure 1.

For each time step from $t = 1$ to $t = n$ ($n \in \mathbb{Z}^*$ denotes the sample size), the FNN applies the following forward propagation equations:

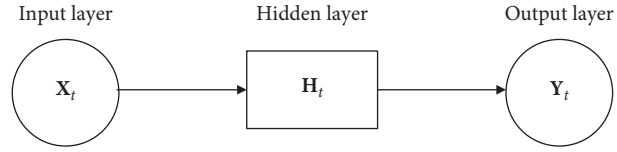

Figure 1: Feedforward neural network.

$$\begin{aligned}\mathbf{H}_t &= f(\mathbf{b} + \mathbf{W}\mathbf{X}_t),\\ \mathbf{Y}_t &= \mathbf{c} + \mathbf{V}\mathbf{H}_t,\\ \widehat{\mathbf{Y}}_t &= g(\mathbf{Y}_t).\end{aligned} \tag{5}$$

The parameters and functions that govern the computation happening in a FNN are described as follows:

(i) $\mathbf{X}_t$ is the input at time step t and is a vector composed of the previous inputs (i.e., $\mathbf{X}_t = [X_{t-k}, \ldots, X_{t-1}]$, where k refers to the step of memory).

(ii) $\mathbf{H}_t$ is the hidden state at time step t, where the bias vectors $\mathbf{b}$ and input-to-hidden weights $\mathbf{W}$ are derived via training. The number of hidden layers and the number of hidden nodes per layer are defined before training, which called hyperparameters in neural networks.

(iii) $\mathbf{Y}_t$ is the output at step t, where the bias vectors $\mathbf{c}$ and hidden-to-output weights $\mathbf{V}$ are derived via training.

(iv) $\widehat{\mathbf{Y}}_t$ is our predictive output at time step t, which would be a vector of probabilities across our sample space.

(v) The function $f(\cdot)$ is a fixed nonlinear function called activation function, and the function $g(\cdot)$ is an output function used in the final layer of a neural network. Both of the two functions belong to hyperparameters which are defined before training (Section 3.2).

The models are called feedforward because the information flows through the approximate function from the input $\mathbf{X}_t$, through the internal computations used to update the model to define $f(\cdot)$, and to the output $\widehat{\mathbf{Y}}_t$ finally. Besides, there is no feedback connection, namely, the outputs of the model are fed back into itself.

2.3.2. RNNs. If adding the feedback connections to the network, then it is called RNNs. In particular, the RNN records the information that has been calculated so far and use it for the calculation of the present output. The principle of RNNs is depicted in Figure 2.

For each time step from $t = 1$ to $t = n$, the RNN applies the following forward propagation equations:

$$\begin{aligned}\mathbf{H}_t &= f(\mathbf{b} + \mathbf{W}\mathbf{H}_{t-1} + \mathbf{U}\mathbf{X}_t),\\ \mathbf{Y}_t &= \mathbf{c} + \mathbf{V}\mathbf{H}_t,\\ \widehat{\mathbf{Y}}_t &= g(\mathbf{Y}_t).\end{aligned} \tag{6}$$

The parameters and functions that govern the computation happening in a RNN are described as follows:

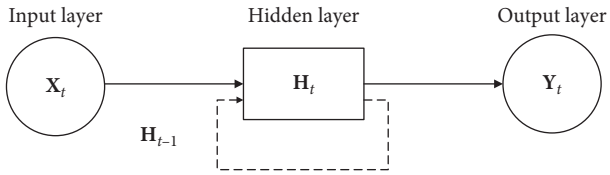

Figure 2: Recurrent neural network.

(i) $\mathbf{X}_t$ is the input at time step t and is one-hot vector. For example, if $X_t = 1$ and the sample space $S = \{0, 1\}$, then $\mathbf{X}_t = [0, 1]$.

(ii) $\mathbf{H}_t$ is the hidden state at time step t. It is the "memory" of the network. $\mathbf{H}_t$ is calculated based on the previous hidden state $\mathbf{H}_{t-1}$ and the input at the current step $\mathbf{X}_t$. $\mathbf{b}$, $\mathbf{U}$, and $\mathbf{W}$ denote the bias vectors, input-to-hidden weights, and hidden-to-hidden connection into the RNN cell, respectively.

(iii) $\mathbf{Y}_t$ is the output at step t. $\mathbf{c}$ and $\mathbf{V}$ denote the bias vectors and hidden-to-output weights, respectively.

(iv) $\widehat{\mathbf{Y}}_t$ is our predictive output at time step t, which would be a vector of probabilities across our sample space.

(v) Similarly, the function $f(\cdot)$ is an activation function and $g(\cdot)$ is an output function, which are defined before training (Section 3.2).

3. Predictors for Min-Entropy Estimation Based on Neural Network

The neural network is able to approximate the various PDFs, and the complexity of neural network is increased slower (linear relationship) as the sample space increases. Motivated by [24], we propose two predictive models based on neural networks for min-entropy estimation. Next, we present the execution strategy of our min-entropy estimators, provide the choices of the important hyperparameters, and give the analysis on the accuracy and complexity of our predictive models to prove that our design is feasible.

3.1. Strategy of Our Predictors for Min-Entropy Estimation. The execution strategy of our min-entropy estimator is depicted in Figure 3, which consists of model training and entropy estimation. Both of our proposed two predictive models (namely predictors), which are based on FNN and RNN, respectively, follow the same strategy.

The benefit of this strategy is that it applies not only to stationary sequences generated by entropy sources but also to nonstationary sequences of which the probability distribution is time-varying. On the one hand, in our strategy, in order that the model can match the statistical behavior of the data source well, we use the whole input dataset to train and continuously update the model. On the other hand, to effectively estimate the entropy of the data source, we use the predictive model to compute the min-entropy only when the predictive model is updated enough to characterize the statistical behavior of the tested dataset. Specifically, for the testing dataset which is used for computing entropy estimation, we preset the testing dataset as a part of the whole observations and utilize a proportion parameter ($\gamma \in [0, 1]$) to determine the size of testing dataset, namely, the last γ of the inputs are used for computing entropy while the model is also updating.

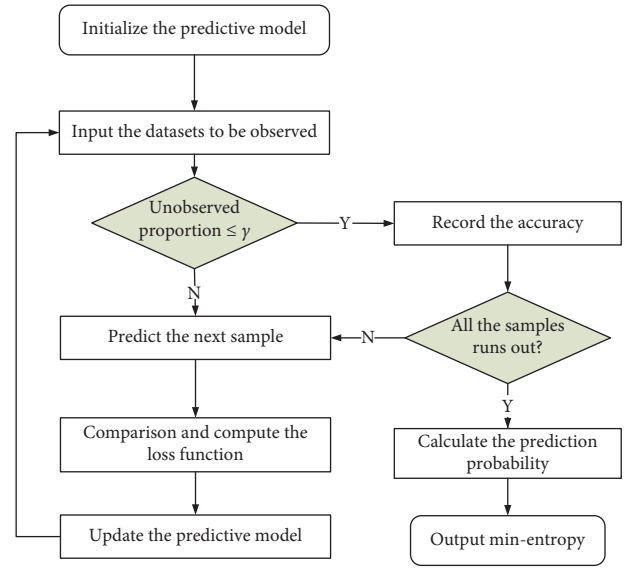

Figure 3: Execution strategy of our predictors for min-entropy estimation based on neural network.

The workflow of the min-entropy estimator based on neural network is listed as the following steps:

(1) Initialization: choose one model (FNN or RNN) and set the hyperparameters of the model.

(2) Input data: input the tested dataset and judge the proportion of the remaining data in the entire tested dataset. If the proportion of the remaining observations $\leq \gamma$, record the accuracy for predicting the next sample to be observed; else, continue.

(3) Prediction: predict the current output according to the forward propagation equations.

(4) Comparison and loss function: observe the real output and compare it to the predicted value. Then, compute the loss function.

(5) Update predictive model: compute the gradient and update the predictive model. If the entire dataset runs out, please turn to Step 6; else, please repeat Step 2 ~ Step 5.

(6) Calculate the predictive accuracy and probability: obtain the accuracy of the executed predictive model from the last γ observations, including global predictions and local predictions, and compute the probability, respectively.

(7) Calculate the min-entropy: calculate the min-entropy by the obtained probability. After obtaining the estimations from the two predictors (FNN and RNN), respectively, the minimum entropy of the two predictors is taken as the final entropy estimation (namely, min-entropy) of the tested dataset.

Combining the calculation principle of min-entropy in Section 2.1, we can see the lower bound on the probability of making a correct prediction gives an upper bound on the entropy of the source. In other words, the more predictable a source is, the larger probability of making correct predictions is, and the less entropy it has. Therefore, a model that is a bad fit for the source or not fully trained will result in inaccurate predictions, a low accurate prediction probability, and a too-high entropy estimation of the source. So, the models that are bad fit for the source or not fully trained can give big overestimates but not underestimates.

Further, we can confirm that adding one more predictor will not do any harm and conversely will make the entropy estimation much more accurate. From the execution strategy, we can see that if all the predictors whose models are not matched for the noise source are used alongside a predictor whose underlying model matches the source's behavior well, then the predictor which matches the source well will determine the final entropy estimation.

3.2. Choices of Important Parameters

3.2.1. Hyperparameters for FNN and RNN. In neural networks, the choices of models' hyperparameters have significant influences on the computational resource and performance required to train and test. Therefore, the choices of hyperparameters are crucial to neural networks. Next, we illustrate the choices of some key hyperparameters.

(1) Hidden Layers and Nodes. Comprehensively balance the accuracy and efficiency of our predictors; in this paper, for the FNN model, except for the multivariate M-sequences, we set the number of hidden layers as 2 and the number of hidden nodes per layer is 10 and 5, respectively. While for the multivariate M-sequences, after extensive tests, the number of hidden nodes per layer shall be larger to give better results. By observing the results, finally we set the numbers as 35 and 30, respectively.

(2) Step of Memory. The step of memory determines the number of previous samples used for predicting the current output. Generally speaking, the larger the value, the better the performance. However, the computational resources (memory and runtime) increase as the step of memory grows. In this paper, we set the step of memory as 20 by trading off performance and resource. That is to say, as for the FNN, the input at time step t is the previous 20 observed values, and as for the RNN, the hidden layer contains 20 unfolded hidden units.

(3) Loss Function. The loss function refers to the function that can measure the difference between the predicted values and the true values. The total loss for a given sequence of $\mathbf{x} = \{x_1, \ldots, x_n\}$ values paired with a sequence of $\mathbf{y} = \{y_1, \ldots, y_n\}$ values would then be just the sum of the losses over all the time steps. For example, if L_t is the negative log-likelihood of y_t given $x_1, \ldots, x_t$, then

$$\begin{aligned} &L(\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_n\}) \\ &= \sum_t L_t \\ &= -\sum_t \log_2\left(p_{\text{model}}\left(y_t \mid x_1, \ldots, x_t\right)\right), \end{aligned} \tag{7}$$

where $p_{\text{model}}(y_t \mid x_1, \ldots, x_t)$ is given by reading the entry for y_t from the model's output vector $\hat{\mathbf{y}}_t$. The models are trained to minimize the cross-entropy between the training data and the models' predictions (i.e., equation (7)), which is equivalent to minimizing the mean squared error (i.e., the average of the squares of the errors or deviations).

(4) Learning Rate. Learning rate is a positive scalar determining the size of the step. To control the effective capacity of the model, we need to set the value of learning rate in an appropriate range. The learning rate determines how fast the parameter θ moves to its optimum value. If the learning rate is too large, gradient descent can inadvertently increase rather than decrease the training error, namely, the parameters are likely to cross the optimal value. However, if the learning rate is too small, the training is not only slower but may become permanently stuck with a high training error. So, the learning rate is crucial to the performance of the model.

Based on the above analysis, we pick the learning rate approximately on a logarithmic scale, i.e., the learning rate taken within the set $\{0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$. At the beginning of model training, we set the learning rate as larger value to faster reach the optimum value. Then, with the number of training increasing, we set the smaller value for not crossing the optimal value. The detailed settings are described in Algorithm 1.

(5) Activation Function. In general, we must use a nonlinear function to describe the features. Most neural networks do so using an affine transformation controlled by learned parameters, followed by a fixed nonlinear function called an activation function. Activation function plays an important role in neural networks. The commonly used activation functions include $\tanh(\cdot)$, $\text{relu}(\cdot)$, and *sigmoid* function which is defined as $\sigma(\cdot)$, i.e., equation (8)) in this paper. Because the *sigmoid* function is easy to saturate, which causes the gradient to change slowly during training, it is generally no longer used as an activation function except in RNN-LSTM (long short term memory):

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{8}$$

After many attempts (i.e., we compare the efficiency and performance by means of the exhaustive method manually), we finally choose the $\tanh(\cdot)$ and $\text{relu}(\cdot)$ as activation functions for FNN and RNN, respectively. They can be expressed as

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{9}$$

Compared with $\sigma(\cdot)$, $\tanh(\cdot)$ is symmetrical about the origin. In some cases, this symmetry can give better

(1) **if** train_num < train_dataset_size/3 **then**
(2) learning rate ⟵ {0.1, 0.01};
(3) **else if** train_num < train_dataset_size/1.5 **then**
(4) learning rate ⟵ $\{0.01, 10^{-3}, 10^{-4}\}$;
(5) **else**
(6) learning rate ⟵ $\{10^{-4}, 10^{-5}\}$;
(7) **end if**

Algorithm 1: Setting of learning rate.

performance. It compresses the real-valued input to a range of −1 to 1, and the mean of its output is zero, which makes it converge faster than the $\sigma(\cdot)$ and reduce the number of iterations. Therefore, this is suitable for activation function, and the zero-centered training dataset contributes the convergence speed of model training:

$$\text{relu}(x) = \max(0, x), \tag{10}$$

where $\text{relu}(\cdot)$ is currently a popular activation function. It is linear and gets the activation value which requires the only one threshold. We choose this function based on the following two considerations. On the one hand, it solves vanishing gradient problem of back propagation through time (BPTT) algorithms for the reason that the derivative of $\text{relu}(\cdot)$ is 1. On the other hand, it greatly improves the speed of calculation because it only needs to judge whether the input is greater than 0.

(6) Output Function. The output function is used in the final layer of a neural network model. The predictors for time series are considered as a solution to a multiclass classification problem, so we take $\text{softmax}(\cdot)$ as the output function, which can be expressed as

$$y_i = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{i=1}^{s} e^{z_i}}, \tag{11}$$

where s is the size of sample space and $\text{softmax}(z_i)$ denotes the probability of the output is z_i and satisfies that $\sum_{i=1}^{s} y_i = 1$, i.e., the sum of the probability of all the outputs is equal to 1. Such networks are commonly trained under a cross-entropy regime (i.e., the loss function mentioned above).

3.2.2. Selection of Testing Dataset Length. To better estimate the entropy of the data source, the length of the testing dataset is very important for min-entropy estimation for random numbers generated by different types of sources. In reality, most entropy sources are time-varying (namely, nonstationary), which means the probability distribution of the output sequences from the source is changing over time. So, the length of the testing dataset shall be adaptive to the type of the source.

Therefore, as described in Section 3.1, we utilize γ to determine the size of testing dataset. Specifically, in our strategy, for the stationary entropy source, of which the probability distribution of the outputs from the source is not changing over time, the parameter γ is preset to 20%. Relatively, for the nonstationary entropy source, all observation points (namely, γ is 100%) need to serve as the testing dataset.

To verify the reasonableness of the γ value, we compute the root-mean-squared error (RMSE) of the lowest estimations of our predictors over 80 sequences from the following simulated datasets generated by nonstationary source:

(i) *Time-Varying Normal Distribution Rounded to Integers.* The samples are subject to a normal distribution and rounded to integer values, but the mean of the distribution moves along a sine curve to simulate a time-varying signal.

The RMSE, i.e., $\sqrt{(1/N)\sum_{i=1}^{N}(\widehat{H}_{\min} - H_{\min})^2}$, refers to the arithmetic square root for the mean of the squares of the errors or deviations for each class of simulated sources. Note that, here N indicates the number of test samples, $\widehat{H}_{\min}$ indicates the estimated result for each sample, and $H_{\min}$ means the theoretical result for each sample. In other words, the smaller the RMSE is, the closer the estimated result is to the theoretical entropy, which indicates the predictor has a better accuracy.

As shown in Table 1, we can see that for the time-varying data source, only when the γ is 100% (namely, the entire dataset shall be used for min-entropy estimation), the predictors can give the most accurate results. This means when the probability distribution of data sources is varying with time, the part of the input dataset cannot represent the overall distribution of the input dataset, so the part of the input dataset cannot accurately give the estimation result of the entire input dataset. Besides, for the stationary sources, it is reasonable that γ is preset to 20% because the estimated results obtained by our method are very close to the correct entropy (in theory) of the selected entropy source as presented in Section 4.1.

3.3. Evaluation on Our Predictors. In this section, we conduct some experiments on simulated datasets to verify the accuracy of our proposed predictors for the min-entropy estimation and compare the experimental results with theoretical results. In addition, we have a theoretical analysis of the complexity of our predictors. Note that, in Section 4, we will apply our predictors to different data sources and provide the comparison on our predictors with 90B's predictors.

3.3.1. Accuracy Verification. We train our predictive models FNN and RNN on a number of representative simulated data sources (including stationary and nonstationary entropy sources), of which the theoretical entropy can be obtained from the known probability distribution of the outputs. Simulated datasets are produced using the following distribution families adopted in [24]:

(1) Simulated Datasets Generated by Stationary Sources.

(i) *Discrete Uniform Distribution.* The samples are equally likely, which come from an IID source.

Table 1: Error measures of final estimations of our predictors for nonstationary sources with different γ values.

γ	0.1	0.2	0.4	0.6	0.8	1
RMSE	0.0911	0.1364	0.0788	0.0817	0.0219	0.0149

(ii) *Discrete Near-Uniform Distribution.* All samples are equally likely except one, which come from an IID source. A certain sample has a higher probability than the rest.

(iii) *Normal Distribution Rounded to Integers.* The samples are subject to a normal distribution and rounded to integer values, which come from an IID source.

(iv) *Markov Model.* The samples are generated using a d^{th}-order Markov model, which come from a non-IID source.

(2) Simulated Datasets Generated by Nonstationary Sources. These datasets are the same as those used in Section 3.2.2.

For every class listed above, we generate a set of 80 simulated datasets, each of which contains 10^6 samples, and estimate min-entropy by using predictive models FNN and RNN, respectively. For each dataset, the theoretical min-entropy $H_{\min}$ is derived from the known probability distribution.

From Figures 4–9, the abscissa in the figure represents the theoretical entropy of the test sample, and the ordinate represents the estimated entropy of the test sample. Figure 4 shows the estimated entropy results for the 80 simulated datasets with uniform and near-uniform distributions, respectively. From Figures 4(a) and 4(b), we see that the estimated results given by our proposed two predictive models (FNN and RNN) are almost consistent with the theoretical entropy for both uniform and near-uniform distributions. So, the final estimated result which is the minimum result of the two predictive models is also basically consistent with the theoretical entropy. Figure 5 shows the estimated entropy results for the 80 simulated datasets with normal distributions and time-varying normal distributions, respectively. From Figures 5(a) and 5(b), we can see that the estimated results given by our proposed two predictive models are close to the theoretical entropy with normal distributions and time-varying normal distributions. According to our execution strategy, here we calculate min-entropy estimations using the whole input dataset for time-varying normal distributions.

Figure 6 shows the estimated results of Markov distributions; we can see that both of our predictive models give a number of overestimates when applied to the Markov sources, particularly with the theoretical entropy increasing.

Table 2 shows the relative errors (namely, $|(\widehat{H}_{\min} - H_{\min})/H_{\min}| * 100\%$) between the theoretical results and the estimated results of FNN and RNN to further reflect the accuracy of the models. $\widehat{H}_{\min}$ and $H_{\min}$ have the same meaning as in Section 3.2.2. We see that the entropy to be estimated with an error of less than 6.02% for FNN and 7% for RNN for the simulated classes, respectively.

Based on the above accuracy verification of our predictors with simulated datasets from different distributions, what we can be sure is that out predictors can give almost accurate results except Markov distributions.

3.3.2. Complexity Analysis. To analyze the usability of our predictors in terms of execution efficiency, we derive the following computational complexity through the analysis of theory and principle of implementation.

We believe that the computational complexity of entropy estimators used for RNG evaluation mainly comes from the sample space and sample size. For ease of analysis, we define the following parameter n as the sample size which indicates the length of the sample, s as the sample space which means the kinds of symbols in the sample (i.e., $s = 8$ means there are 8 symbols in the sample, and the bit width of each symbol is $\log_2(8) = 3$, such as 010, 110, 111, ...), and k denotes the maximum step of correlation which is set as a constant in 90B's predictors ($k = 16$) and our predictors ($k = 20$).

Through the analysis of the implementation, the computational complexity of the final 90B's predictors [23] mainly comes from the MultiMMC predictor and is of order $O(s^k \cdot n + 2^k \cdot n \cdot \log_2(s))$, which is mainly linear time complexity of n and k-order polynomial time complexity of s. While the computational complexity of our predictor is of order $O(s \cdot n)$, which is linear time complexity of s and n. It can be seen that the computational complexity of our predictors is much lower than that of the 90B's predictors.

It is important to note that the MultiMMC predictor requires $s^k \ll n$; otherwise, this predictor cannot give accurate estimated results statistically. That is to say, when the s is increasing, the MultiMMC predictor requires larger sample size in order to estimate the entropy accurately.

From the above analysis, we can see our predictors have lower computational complexity. We will give the experimental proof in Section 4.3.

4. Comparison on Our Predictors With 90B'S Predictors

In this section, a large number of experiments have been done to evaluate our proposed predictors for entropy estimation from the aspects of accuracy, applicability, and efficiency by applying our predictors to different simulated data and real-world data. For the experiments mentioned above, we compare the results with the final 90B's predictors [23] to highlight the advantages of our work. Similarly, our predictors in these experiments compute an upper-bound of min-entropy estimation at the significance level $\alpha = 0.01$ which is the same as 90B's predictors.

4.1. Comparison on Accuracy

4.1.1. Simulated Data. The simulated datasets are produced using the same distribution families as described in Section 3.3.1. Further, we append the following two new distribution families, such as pseudorandom sequence and post-processing sequence which are representative and commonly used in reality:

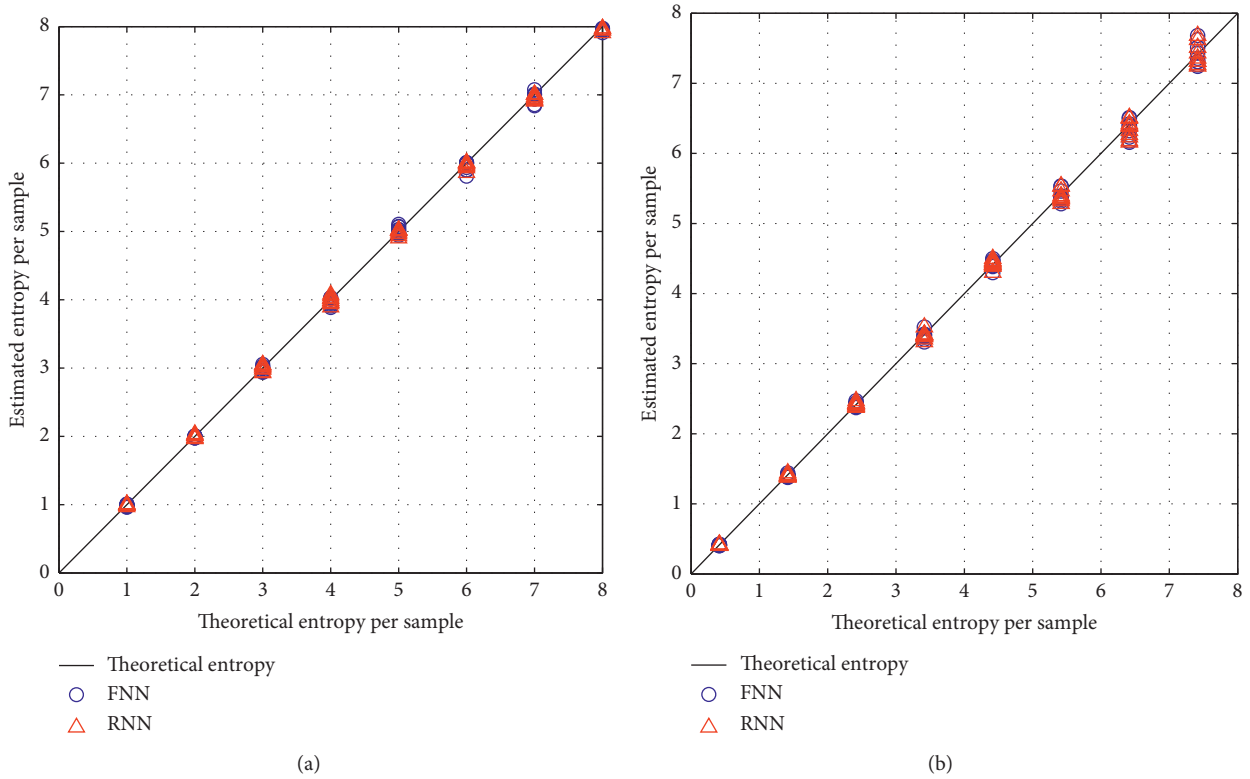

Figure 4: Comparison of estimated results obtained from our two predictive models with the theoretical entropy. Estimations for (a) uniform distributions and (b) near-uniform distributions.

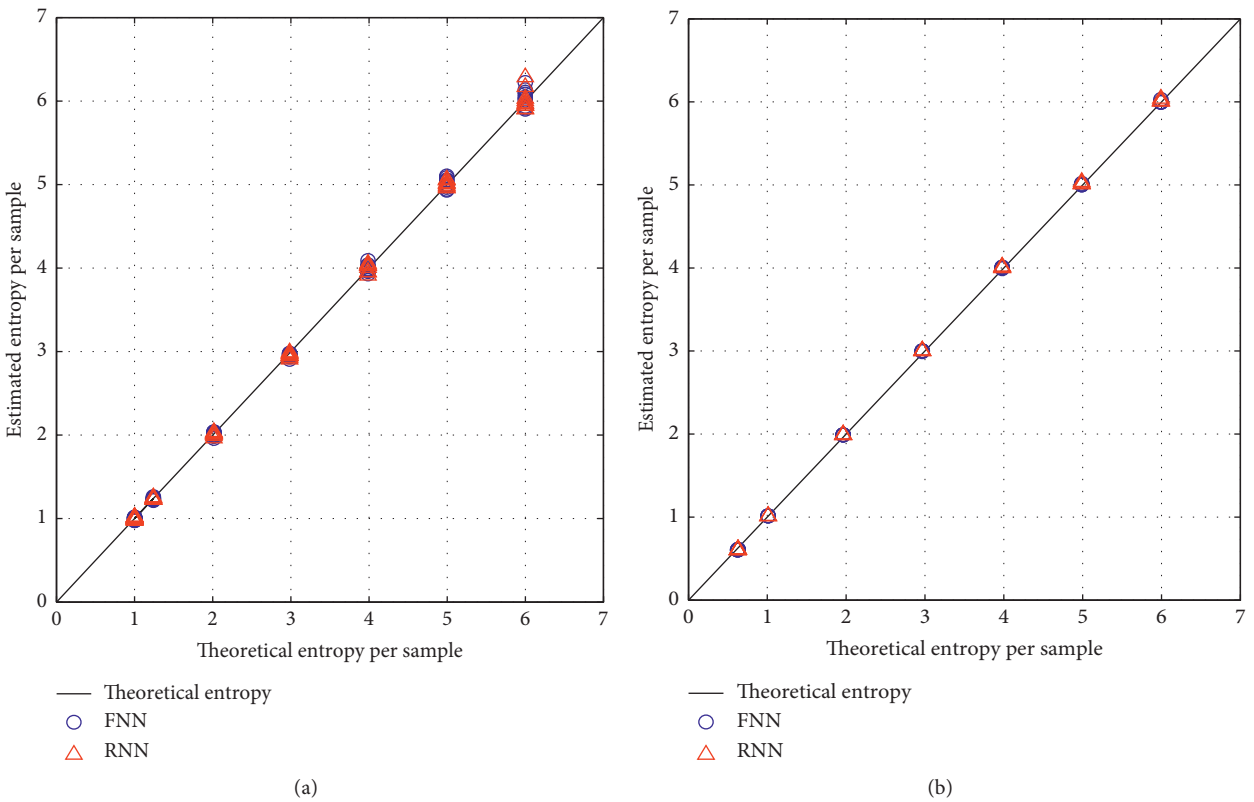

Figure 5: Comparison of estimated results obtained from our two predictive models with the theoretical entropy. Estimations for (a) normal distributions and (b) time-varying normal distributions.

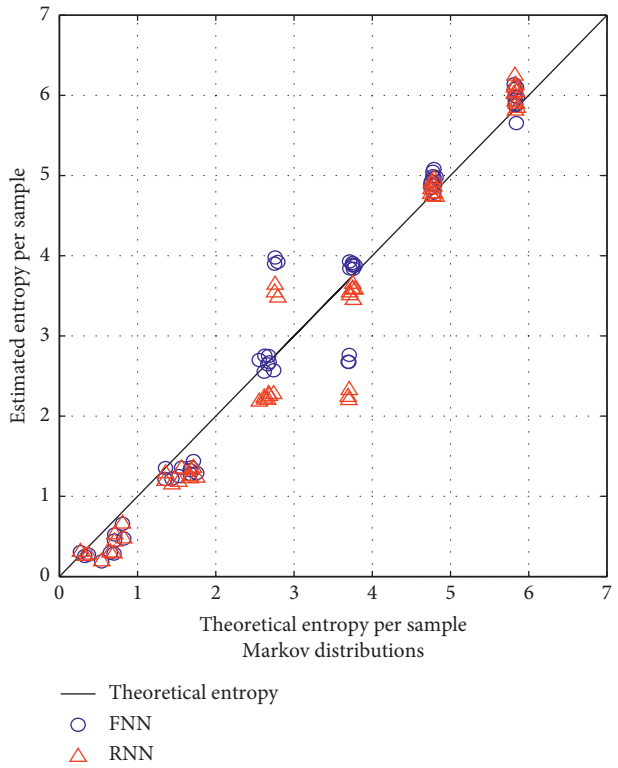

Figure 6: Comparison of estimated results obtained from our two predictive models with the theoretical entropy for Markov distributions.

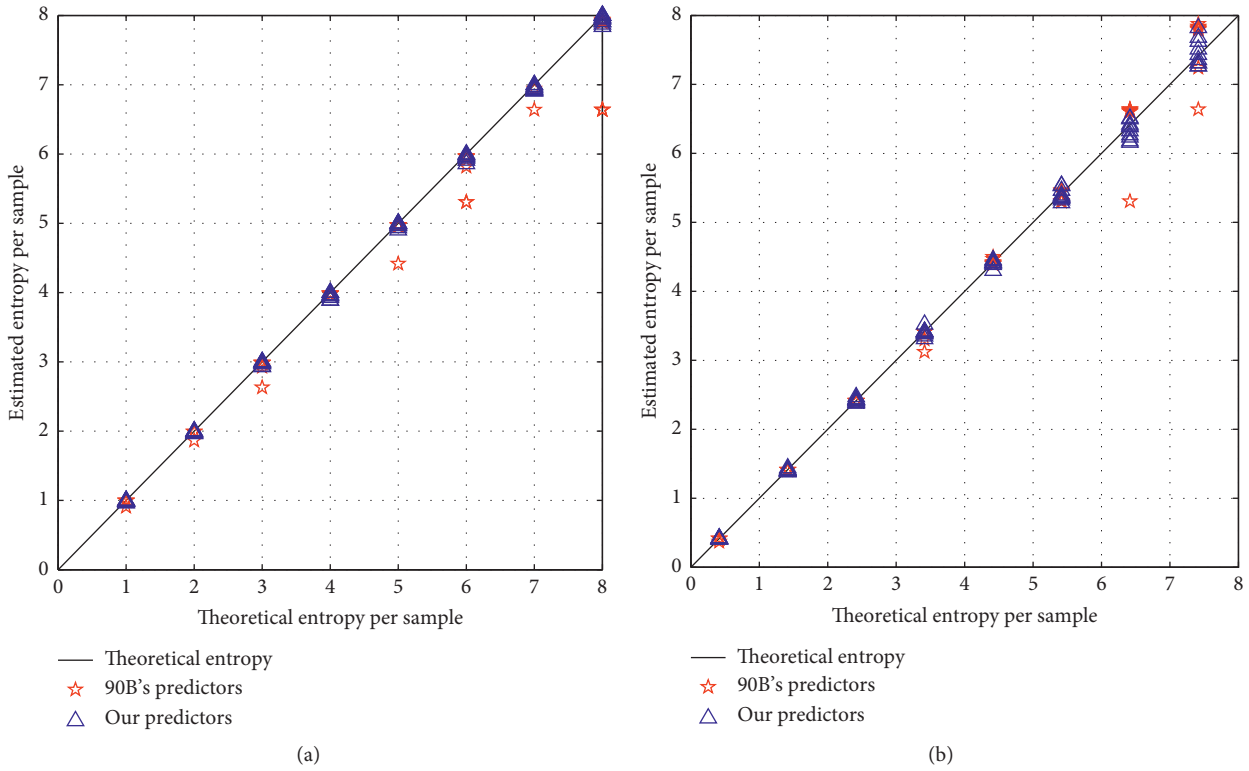

Figure 7: Comparison of the min-entropy estimation obtained from our proposed predictors and 90B's predictors with the theoretical entropy. Estimations for (a) uniform distributions and (b) near-uniform distributions.

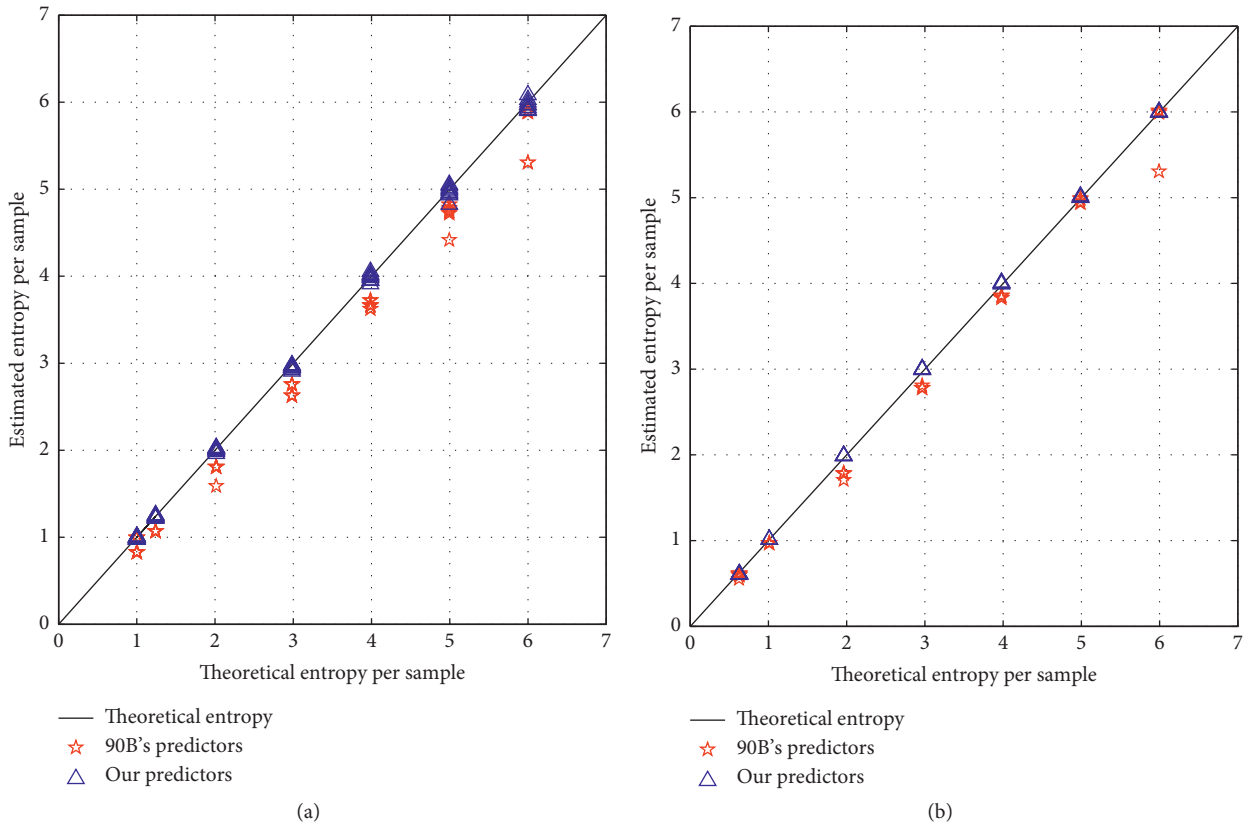

Figure 8: Comparison of the min-entropy estimations obtained from our proposed predictors and 90B's predictors with the theoretical entropy. Estimations for (a) normal distributions and (b) time-varying normal distributions.

(i) *M-Sequence.* A maximum length sequence, which is a type of pseudorandom binary sequence ([35])

(ii) *Nonuniform Distribution by Postprocessing Using LFSR.* The samples are processed using a linear feedback shifting register (LFSR), which come from an IID source ([35]).

For every distribution mentioned above, we also generate a set of 80 simulated datasets, each of which contains 10^6 samples, and estimate min-entropy by using our proposed predictors and final 90B's predictors [23].

Figure 7 shows the estimated min-entropy results for the 80 simulated datasets with uniform distributions and near-uniform distributions, respectively. From Figures 7(a) and 7(b), we see that several points of the results obtained from the 90B's predictors are apparently underestimated, which may result from the overfitting phenomenon. Compared with 90B's predictors, our predictors provide more accurate results.

Figure 8 shows the estimated min-entropy results for normal distributions and time-varying normal distributions, respectively. From Figures 8(a) and 8(b), we can see that the estimated results given by our predictors are close to the theoretical entropy with normal distributions and time-varying normal distributions. However, the lowest entropy estimation results obtained from the 90B's predictors give significant underestimates.

Figure 9 shows the estimated min-entropy results for Markov distributions. We can see that the 90B's predictors almost give underestimates compared with the theoretical entropy, while estimated results given by our predictors are much closer to the theoretical entropy than those obtained from 90B's predictors.

To further obviously compare the accuracy of our and 90B's predictors, we apply the predictors to the M-sequence and the non-uniform distribution sequence by post-processing using LFSR, and their theoretical entropy is a known and fixed value.

It is further confirmed that the higher stage (the maximum step of correlation) M-sequence and nonuniform distribution sequence by postprocessing using LFSR are able to pass the NIST SP 800-22 statistical tests [8]. The estimated results are listed in Tables 3 and 4, and the lowest entropy estimations from 90B's predictors and our predictors for each stage are shown in bold font.

For M-sequence and nonuniform distribution by post-processing using LFSR, the MultiMMC predictor presented in the final 90B gives the most accurate entropy estimation results for the stage ≤ 16. However, when the stage of M-sequence and nonuniform distribution by postprocessing using LFSR is greater than 16, the MultiMMC predictor cannot give accurate entropy estimation result because this predictor is parameterized by $k \in \{1, 2, \ldots, 16\}$ (k is the maximum step of correlation). Perhaps, we could set the

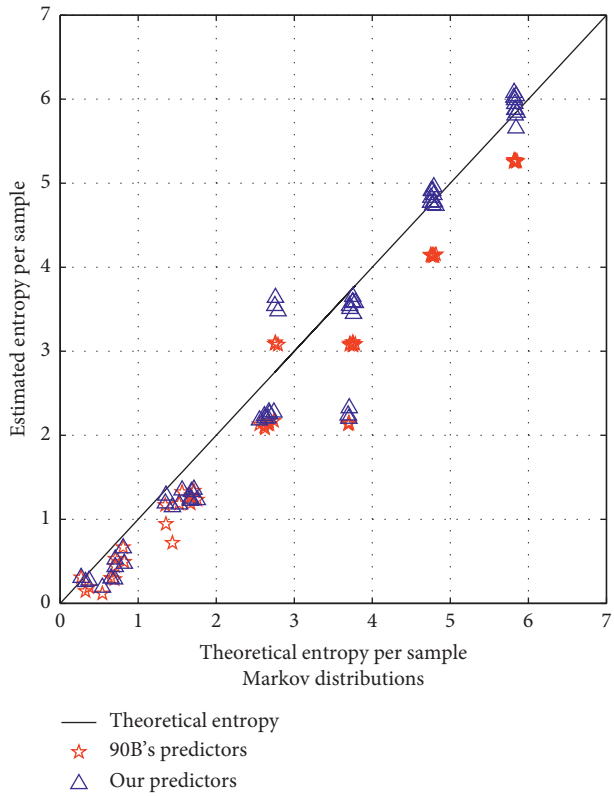

Figure 9: Comparison of the min-entropy estimations obtained from our proposed predictors and 90B's predictors with the theoretical entropy for Markov distributions.

Table 2: Relative errors of FNN and RNN estimation results.

Simulated data class	FNN (%)	RNN (%)
Uniform	1.65	1.6
Near-uniform	1.60	1.52
Normal	1.17	1.08
Time-varying normal	2.12	1.84
Markov	6.02	7

Table 3: Estimated results for M-sequence ($H_{\min} = 0.000$).

Stage	8	10	12	14	16	18	20
MultiMCW	0.991	0.996	0.988	0.989	0.993	**0.999** *	1.000
Lag	1.000	1.000	1.000	1.000	1.000	1.000	1.000
MultiMMC	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	1.000	1.000
LZ78Y	1.000	1.000	1.000	1.000	1.000	1.000	**0.997** *
FNN	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
RNN	0.000	1.048	1.007	1.002	0.996	0.9920	0.9997

parameter of the MultiMMC predictor as a greater range to achieve a more accurate estimated result for the higher stage, but the time complexity grows exponentially with the parameter k as we analyzed in Section 3.3.2. Moreover, the FNN model can also give accurate estimated results, even though the stages of M-sequence and LFSR are greater than 16. However, the RNN model can give accurate estimated results only when the stage is 8. Therefore, the FNN model is more matched to M-sequence and nonuniform distribution by postprocessing using LFSR than RNN.

Table 4: Estimated results for nonuniform distribution by postprocessing using LFSR ($H_{\min} = 0.152$).

Stage	8	10	12	14	16	18	20
MultiMCW	0.440	0.595	0.743	0.721	0.998	0.994	0.998
Lag	0.581	0.581	0.680	0.680	0.992	0.994	0.999
MultiMMC	**0.151**	**0.153**	**0.158**	**0.181**	**0.234**	0.995	0.996
LZ78Y	0.567	0.995	0.766	0.679	0.997	**0.996** *	**0.994** *
FNN	**0.151**	**0.145**	**0.149**	**0.147**	**0.149**	**0.142**	**0.144**
RNN	0.149	0.947	1.012	0.998	1.012	0.997	0.985

We also compute the relative errors of estimated results from 90B's predictors and our predictors over 80 sequences from each class of simulated sources. We calculate the relative errors using the min-entropy obtained from 90B's predictors (the lowest estimation result of 90B's four predictors) and our predictors (the lowest estimation result of FNN and RNN), respectively. As illustrated in Table 5, it shows that, for all five classes of simulated sources, the errors of our predictors are lower than that of the 90B's predictors. Specially, our approaches enable the entropy to be estimated with an error of less than 6.55%, but it is up to 14.65% for 90B's predictors. Overall, this indicates that our proposed predictors have a better performance than that of 90B's predictors on accuracy for both stationary sequences and nonstationary sequences, which is consistent with the conclusion drawn in the figures above.

From Tables 2–4, we also find that the accuracy of the RNN predictive model is slightly higher than that of the FNN predictive model, except for the cases of the Markov sources, M-sequence, and nonuniform distribution by postprocessing using LFSR.

We will further verify the applicability for time-varying sources in Section 4.2. Therefore, through the evaluation on the entropy estimation results of the above simulated datasets, we see that our proposed predictors are superior in accuracy compared with the 90B's predictors.

4.1.2. Real-World Data. We further apply our predictors to the datasets which are generated from the RNGs deployed in the real-world. In fact, the theoretical entropy of per sample is unknown for these real-world sources, so no error can be compared like the simulated datasets for the predictors. However, the estimated results from the predictors presented here can still be compared to the 90B's predictors, based on the knowledge that underestimates from the predictors have theoretical bounds.

Datasets of real-world data are produced using the following approaches. The first two are adopted in [24], and the others are commonly used typical RNGs. The estimations of the real-world sources are presented in Table 6.

(i) *RANDOM.ORG.* This is a service that provides random numbers based on atmospheric noise and is used in [24]. It allows the user to specify the

Table 5: Relative errors of the final estimations of 90B's predictors and our predictors for five classes of simulated sources.

Simulated data class	90B's predictors (%)	Our predictors (%)
Uniform	4.37	1.53
Near-uniform	3.47	1.59
Normal	6.08	1.57
Time-varying normal	3.47	1.72
Markov	14.65	6.55

minimum and maximum values that are output. The sequence used here consists of bits.

(ii) *Ubld.it TrueRNGpro.* TrueRNGpro provides a steady stream of random numbers through a USB CDC serial port, which is a USB random number generator produced by Ubld.it. This entropy source is also used in [24]. The sequence used here consists of bits.

(iii) *Linux kernel entropy source.* The Linux kernel random generator is used for the generation of a real-world sequence without any processing. The sequence used here is the last bit of per symbol.

(iv) *Linux/dev/urandom.* The/dev/urandom [6] of Linux is used for the generation of a real-world sequence with strict processing. The sequence used here consists of bits.

(v) *Windows RNG.* Windows RNG [5] is used for the generation of a real-world sequence by calling a Crypto API. The sequence used here consists of bits.

As illustrated in Table 6, the lowest entropy estimation for each source is shown in bold font. We see that our predictors perform better than 90B's predictors, because the lowest entropy estimation is always obtained from our work for each real-world source. Furthermore, for Linux kernel entropy source, we find that both of the predictor Lag and MultiMMC are able to give lower estimation results. It indicates that Linux kernel entropy source has periodicity and conforms to the Markov model, which is well understood because the randomness of Linux kernel entropy source comes from human behaviors, such as manipulating the mouse and keyboard. In our work, compared with the entropy estimations for other real-world sources, FNN fits much better than RNN for Linux kernel entropy source, which is consistent with the previous view that FNN performs well in testing Markov sources.

4.2. Comparison on the Scope of Applicability. After evaluating the accuracy, we further validate the scope of applicability of our proposed predictors and compare them with that of the 90B's predictors. Kelsey et al. [24] stated that each of the 90B's predictors performs well only for a special distribution as described in Section 2.2.1. To prove our predictor has better applicability, the following four simulated datasets are generated, which are suitable for each predictor employed in the final 90B:

(i) *Time-Varying Sources.* The probability distribution of data sources is varying with time. The MCW predictor predicts the current output according to previous outputs in a short period of time, and thus the MCW predictor performs well in these data sources.

(ii) *Periodic Sources.* The data source changes periodically. The lag predictor predicts the value that occurred samples back in the sequence as the current output, and thus the lag predictor performs well on sources with strong periodic behavior.

(iii) *Markov Sources.* The data sources can be modeled by the Markov model. The MultiMMC predictor predicts the current output according to the Markov model, and thus the MultiMMC predictor performs well on data from any process that can be accurately modeled by a Markov model.

(iv) *LZ78Y Sources.* The data sources can be efficiently compressed by LZ78-like compression algorithms, which applies to the LZ78Y predictor well.

For each above simulated source, we generate a set of 10 simulated datasets, each of which contains 10^6 samples, and the min-entropy is estimated by our and 90B's predictors. The final result for a predictor is the average value of 10 estimated results corresponding to the 10 simulated datasets for one simulated source.

4.2.1. Time-Varying Sources. Firstly, we generate the time-varying binary data which is suitable for the statistical behaviors of the MCW predictor presented in the 90B. Table 7 shows the entropy estimation results for time-varying data.

As shown in Table 7, symbol *gradual*(x) ($x \in [0, 1]$, the same below) is defined as a simulated source that the probability of output "0" changes gradually from x to $1 - x$ with time. Symbol *period*(x) is defined as a simulated source that the probability of output "0" changes periodically with time, and the probability varies from x to $1 - x$ in one period. The period length is set to 20% of the entire input dataset. Symbol *sudden*(x) is defined as a simulated source that the probability of output "0" changes suddenly with time, namely, the probability is set to x for the first half of the input dataset and $1 - x$ for the last half.

From Table 7, the estimation results for MCW predictor and our work are shown in bold font. We see that the MCW predictor gives the lowest and most accurate entropy estimations for the three types of time-varying data mentioned above, but it gives a little underestimates at gradual(0.2) and period(0.2). It is confirmed that the time-varying sources mentioned above match with the statistical behaviors of the MCW predictor. Relatively, we find that our proposed predictive models are all capable to obtain the satisfied entropy estimations that are close to the correct values. Therefore, it is proved that our proposed predictive models are suitable for the time-varying data mentioned above. Note that we calculate the min-entropy estimate according to the entire dataset rather than the last 20% of the input dataset for these time-varying sources. Because the probability distribution is varying with time, the part of the input dataset cannot represent the overall distribution of the input dataset.

Table 6: Entropy estimates for real-world sources.

Real-world sources	90B's predictors				Our work	
	MCW	Lag	MultiMMC	LZ78Y	FNN	RNN
RANDOM.ORG	**0.9951**	0.9963	0.9966	0.9976	**0.9802**	0.9954
Ubld.it TrueRNGpro	0.9979	**0.9955**	0.9973	0.9966	0.9934	**0.9728**
Linux kernel entropy source	0.6173	**0.1232**	0.1269	0.6164	**0.1230**	0.3068
Linux/dev/urandom	0.9952	**0.9935**	0.9990	0.9964	0.9983	**0.9911**
Windows RNG	**0.9953**	0.9986	0.9975	0.9984	**0.9833**	0.9853

Table 7: Entropy estimates for time-varying data.

Data class	Correct	90B's predictors				Our work	
		MCW	Lag	MultiMMC	LZ78Y	FNN	RNN
gradual(0.2)	0.6345	**0.5290**	0.7808	0.7240	0.7790	**0.6288**	0.6289
gradual(0.3)	0.7437	**0.7378**	0.9221	0.8416	0.9243	**0.7430**	0.7460
gradual(0.4)	0.8645	**0.8631**	0.9786	0.9518	0.9739	0.8648	**0.8637**
period(0.2)	0.6345	**0.5537**	0.7428	0.5537	0.7669	**0.6205**	0.6209
period(0.3)	0.7437	**0.7393**	0.9218	0.8476	0.9233	0.7377	**0.7375**
period(0.4)	0.8645	**0.8639**	0.9767	0.9632	0.9796	0.8653	**0.8632**
sudden(0.2)	0.3219	**0.3203**	0.4663	0.3386	0.4484	**0.3217**	0.3229
sudden(0.3)	0.5146	**0.5110**	0.5857	0.9984	0.7663	**0.5110**	0.5119
sudden(0.4)	0.7370	**0.7338**	0.8699	0.9984	0.9389	**0.7339**	0.7345

4.2.2. Periodic Sources. Secondly, we generate periodic data which are suitable for the statistical behaviors of the lag predictor presented in 90B. The following is entropy estimation results for periodic sequences. The data source is completely obeying the periodic rule, so the correct entropy is zero. The bit width of samples is traversed from 2 to 8.

As shown in Table 8, the estimation results for the lag predictor and our work are shown in bold font. According to the correct entropy (is equal to 0) of the simulated periodic sources, we confirm that the lag predictor is suitable for the entropy estimation of this type of source as expected. Relatively, the RNN can also give the accurate min-entropy estimates, i.e., estimated results are zeros. Thus, our proposed predictive models are suitable for the entropy estimation of the (strong) periodic data. In addition, the MultiMMC predictor can also give the accurate min-entropy estimations. This is reasonable because periodicity is also a form of correlation.

4.2.3. Markov Sources. Next, we generate multivariate M-sequences as Markov sources which fit the statistical behaviors of the MultiMMC predictor. Specifically, the multivariate M-sequences are composed of multiple M-sequences with different initial states. Due to the determinacy of this type of sequences, the correct entropy is zero. The bit width of the samples is also traversed from 2 to 8. The maximum step of correlation used here is set as 8. Table 9 shows the estimated results for multivariate M-sequences.

From Table 9, the estimation results for MultiMMC predictor and our work are shown in bold font. According to the correct entropy (is equal to 0) of the simulated Markov sources, we confirm that the MultiMMC predictor is suitable for the entropy estimation of this type of source as expected. Relatively, the RNN can also give the accurate min-entropy estimations, i.e., estimated results are zeros. Thus, our proposed predictive models are suitable for the Markov sources.

Table 8: Entropy estimates for periodic sequences.

Bit width	90B's predictors				Our work	
	MCW	Lag	MultiMMC	LZ78Y	FNN	RNN
2	1.6458	**0.0000**	0.0000	1.1817	0.0079	**0.0000**
3	2.3318	**0.0000**	0.0000	1.5957	0.1315	**0.0000**
4	2.9147	**0.0000**	0.0000	1.8016	0.4748	**0.0000**
5	3.3269	**0.0000**	0.0000	1.4586	0.8898	**0.0000**
6	3.9092	**0.0000**	0.0000	***0.8322***	3.4944	**0.0000**
7	4.4908	**0.0000**	0.0000	***0.3973***	3.4960	**0.0000**
8	4.4919	**0.0000**	0.0000	***0.2027***	3.5408	**0.0000**

*The result with italic font is used to analyze the applicability for the LZ78Y sources.

4.2.4. LZ78Y Sources. Finally, we verify the applicability of the LZ78Y sources. This type of entropy source is difficult to generate by simulating. However, we can still draw the conclusion that our proposed predictive models can be applied to the LZ78Y sources according to Tables 8 and 9 in italic font. Because the periodic data and Markov sequences are compressible.

4.2.5. Summary on Applicability Scope of Our Predictors. By analyzing the experimental results of the above four specific simulated sources, each of which is oriented towards a certain predictor in the 90B, we have a conclusion that our predictors can provide accurate estimated results of entropy. So, the proposed predictors are well applied to these entropy sources as well as the 90B's predictors. In addition, compared with 90B's predictors, our predictors have a better performance on the scope of applicability for testing the

Table 9: Entropy estimates for multivariate M-sequences.

Bit width	90B's predictors				Our work	
	MCW	Lag	MultiMMC	LZ78Y	FNN	RNN
2	1.9010	2.0000	**0.0000**	2.0000	0.0005	**0.0000**
3	2.9906	3.0000	**0.0000**	2.4940	0.0000	**0.0000**
4	3.4037	4.0000	**0.0000**	4.0000	0.0021	**0.0000**
5	4.9753	5.0000	**0.0000**	**1.2269**	0.0041	**0.0000**
6	5.3916	6.0000	**0.0000**	**1.2905**	0.0394	**0.0000**
7	5.3916	6.0000	**0.0000**	**1.9881**	0.0280	**0.0000**
8	7.0000	7.0000	**0.0000**	**0.6611**	0.8635	**0.0000**

*The result with italic font is used to analyze the applicability for the LZ78Y sources.

datasets with long-range correlation as presented in Section 4.1.1.

4.3. Comparison on Execution Efficiency. We implement our predictors and the final 90B's predictors using Python 3.6, and the version of TensorFlow is 1.31.1. All the following tests are conducted on a computer with Intel Core i7 CPU and 32 GB RAM.

Table 10 shows the mean execution time of our predictors in comparison with that of the final 90B's predictors and the second draft of 90B's predictors. Each experimental result in Table 10 is the average value obtained from 50 repeated experiments. Note that the definitions of parameter n, s, and k are the same as in Section 3.3.2.

From the listed mean execution time with different scales ($\{n, s\}$) in Table 10, it can be seen that when $n = 10^6$, the mean execution time of our predictors is much lower and increasing slower with any s than that of the final 90B's predictors. In other words, the average execution efficiency of our predictors is about 7 to 10 times higher than that of the final 90B's predictors for different sample space s when the sample size n is 10^6. In particular, when $n = 10^8$, the mean execution time given by final 90B's predictors is far more than our predictors regardless of the size of sample space and is too long (over three days) to calculate the estimated results on the case $s \geq 2^2$.

In terms of execution efficiency of 90B's predictors, we also find that the mean execution time of the final 90B's predictors is much higher than that of the second draft of 90B's predictors. Actually, the final 90B's mean execution time is about twice as much as the second draft of 90B's. This could be caused by the characteristics of some estimators which are limited to only for binary inputs. Because the collision estimator, Markov estimator and compression estimator are only suitable for binary input (0 or 1) as stated in [23]. So for nonbinary inputs, the 90B's estimators will not only calculate the original symbol entropy but also convert it into binary input to calculate the bit entropy and finally get the min-entropy. This will greatly increase the mean execution time.

4.4. General Discussion. For the most entropy sources which have been tested, the RNN gives more accurate estimations than the FNN. Better accuracy of the RNN predictive model may be due to the following reasons. On the one hand, RNN adds the feedback connections to the network, i.e., it considers not only the relationship between the current output and the previous observations but also the relationship among the previous observations. On the other hand, RNN one-hot-encodes the training dataset for better forecasting categorical data. On the contrary, for Markov sources, M-sequence and nonuniform distribution by postprocessing using LFSR, the current output is only related to the previous observations, which fits the FNN predictive model well and thus the FNN provides more accurate estimated results.

Table 10: Comparison on execution efficiency of min-entropy estimation of our study and 90B's predictors.

$\{n, s\}$	Final 90B (s)	Old 90B (s)	Our predictors (s)
$\{10^6, 2^1\}$	921	561	136
$\{10^6, 2^2\}$	1,058	525	138
$\{10^6, 2^3\}$	1,109	574	149
$\{10^6, 2^4\}$	1,235	598	174
$\{10^6, 2^5\}$	1,394	630	190
$\{10^6, 2^6\}$	1,683	785	186
$\{10^6, 2^7\}$	2,077	938	264
$\{10^6, 2^8\}$	2,618	1,298	272
$\{10^8, 2^1\}$	52,274	47,936	9,184
$\{10^8, 2^2\}$	—	—	9,309
$\{10^8, 2^3\}$	—	—	9,385
$\{10^8, 2^4\}$	—	—	9,836
$\{10^8, 2^5\}$	—	—	10,986
$\{10^8, 2^6\}$	—	—	13,303
$\{10^8, 2^7\}$	—	—	17,649
$\{10^8, 2^8\}$	—	—	20,759

5. Conclusions and Future Work

Entropy estimation provides a crucial evaluation for the security of RNGs. The predictor serves as a universal sanity check for entropy estimation. In this work, we provide several new approaches to estimate the min-entropy for entropy sources using predictors based on neural networks (i.e., FNN and RNN) for the first time. In particular, we design a novel scheme for the proposed entropy estimation based on neural network models, including execution strategy and parameter settings. In order to evaluate the quality of the proposed predictors, we collect various types of simulated sources that belong to the stationary or nonstationary, whose correct entropy of the source can be derived from the known probability distribution, and the theoretical result is further verified by the experiments of the real-world sources. We also compare our method with the predictors defined in the NIST 800-90B (published in 2018) which is a commonly used standard for evaluating the validation of entropy sources. Our assessment experiments are carried out in three aspects, namely, accuracy, scope of applicability, and computational complexity. The experimental results demonstrate that the entropy estimation obtained from our proposed predictors are more accurate than that of the 90B's predictors, and our predictors have a remarkably wide scope of applicability. In addition, the

computational complexity of ours is obviously lower than that of the 90B's with the growing sample space and sample size in theoretical analysis. The average execution efficiency of our predictors is about 7 to 10 times higher than that of the 90B's predictors for different sample spaces when the sample size is 10^6. Specially, the 90B's predictors cannot calculate out a result due to the huge time complexity when the sample space s is over 2^2 with the parameter of maximum step $k = 16$ and sample size $n = 10^8$; relatively, our method is able to provide a satisfied result towards the entropy sources with large sample space and long dependence.

Future work is aiming at designing some specific neural network predictive models for min-entropy estimation for some specific entropy sources. Our future work will also focus on applying this new method to estimate entropy for more application areas, like the randomness sources (sensors and other sources) in mobile terminals.

Data Availability

RANDOM.ORG data used to support the findings of this study can be accessed from https://www.random.org. Ubld.it TrueRNGpro, Linux kernel entropy source and Linux/dev/urandom, and Windows RNG data used to support the findings of this study can be obtained from the relevant listed references.

Disclosure

A preliminary version of this paper appeared under the title "Neural Network Based Min-entropy Estimation for Random Number Generators" in Proc. Security and Privacy in Communication Networks-14th EAI International Conference, SecureComm 2018, Singapore, August 8–10, 2018 [36]. Dr. Jing Yang participated in this work when she studied in Chinese Academy of Sciences, and now she works in China Information Technology Security Evaluation Center, Beijing, China.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Key R&D Program of China (No. 2018YFB0804300), National Natural Science Foundation of China (Nos. 61872357 and 61802396), and National Cryptography Development Fund of China (No. MMJJ20180113).

References

[1] I. Kanter, Y. Aviad, I. Reidler, E. Cohen, and M. Rosenbluh, "An optical ultrafast random bit generator," *Nature Photonics*, vol. 4, no. 1, pp. 58–61, 2010.

[2] P. Li, A. Wang, Y. Guo et al., "Ultrafast fully photonic random bit generator," *Journal of Lightwave Technology*, vol. 36, no. 12, pp. 2531–2540, 2018.

[3] P. Li, K. Li, X. Guo et al., "Parallel optical random bit generator," *Optics Letters*, vol. 44, no. 10, pp. 2446–2449, 2019.

[4] A. Uchida, K. Amano, M. Inoue et al., "Fast physical random bit generation with chaotic semiconductor lasers," *Nature Photonics*, vol. 2, no. 12, pp. 728–732, 2008.

[5] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the random number generator of the windows operating system," *ACM Transactions on Information and System Security*, vol. 13, no. 1, pp. 1–32, 2009.

[6] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the linux random number generator," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P 2006)*, pp. 371–385, Berkeley, CA, USA, May 2006.

[7] M. Vanhoef and F. Piessens, "Predicting, decrypting, and abusing WPA2/802.11 group keys," in *Proceedings of the 25th USENIX Security Symposium*, pp. 673–688, Austin, TX, USA, August 2016.

[8] A. L. Rukhin, J. Soto, J. R. Nechvatal et al., *Sp 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication, Gaithersburg, MD, USA, 2010.

[9] W. Killmann and W. Schindler, *AIS 31: Functionality Classes and Evaluation Methodology for True (Physical) Random Number Generators Version 3.1*, T-Systems GEI GmbH and Bundesamt fur Sicherheit in der Informationstechnik (BSI), Bonn, Germany, 2001.

[10] G. Marsaglia, "The marsaglia random number CDROM including the diehard battery of tests of randomness," 1996, http://www.stat.fsu.edu/pub/diehard/.

[11] P. L'Ecuyer and R. J. Simard, "TestU01: a C library for empirical testing of random number generators," *ACM Transaction on Mathematical Software*, vol. 33, no. 4, 2007.

[12] ISO/IEC JTC 1/SC 27, Berlin, Germany: ISO/IEC 18031: Information Technology–Security Techniques–Random bit Generation, 2011.

[13] E. Barker and J. Kelsey, "Nist draft special publication 800-90B: recommendation for the entropy sources used for random bit generation," 2012, http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf.

[14] M. Baudet, D. Lubicz, J. Micolod, and A. Tassiaux, "On the security of oscillator-based random number generators," *Journal of Cryptology*, vol. 24, no. 2, pp. 398–425, 2011.

[15] W. Killmann and W. Schindler, "A design for a physical RNG with robust entropy estimators," in *Proceedings of the 10th International Workshop Cryptographic Hardware and Embedded Systems—CHES 2008*, pp. 146–163, Washington, D.C., USA, August 2008.

[16] Y. Ma, J. Lin, T. Chen, C. Xu, Z. Liu, and J. Jing, "Entropy evaluation for oscillator-based true random number generators," in *Proceedings of the 16th International Workshop Cryptographic Hardware and Embedded Systems—CHES 2014*, pp. 544–561, Busan, South Korea, September 2014.

[17] Y. Ma, J. Lin, and J. Jing, "On the entropy of oscillator-based true random number generators," in *Proceedings of the Cryptographers' Track at the RSA Conference*, pp. 165–180, Springer, San Francisco, CA, USA, February 2017.

[18] P. Li, J. Zhang, L. Sang et al., "Real-time online photonic random number generation," *Optics Letters*, vol. 42, no. 14, pp. 2699–2702, 2017.

[19] X. Ma, F. Xu, H. Xu, X. Tan, B. Qi, and H. K. Lo, "Post-processing for quantum random-number generators: entropy evaluation and randomness extraction," *Physical Review A*, vol. 87, no. 6, pp. 062327:1–062327:10, 2013.

[20] K. Ugajin, Y. Terashima, K. Iwakawa et al., "Real-time fast physical random number generator with a photonic

integrated circuit," *Optics Express*, vol. 25, no. 6, pp. 6511–6523, 2017.

[21] F. Xu, B. Qi, X. Ma, H. Xu, H. Zheng, and H.-K. Lo, "Ultrafast quantum random number generation based on quantum phase fluctuations," *Optics Express*, vol. 20, no. 11, pp. 12366–12377, 2012.

[22] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "(Second draft) NIST special publication 800-90b: recommendation for the entropy sources used for random bit generation," 2016, https://csrc.nist.gov/CSRC/media/Publications/sp/800-90b/draft/documents/sp800-90b_second_draft.pdf.

[23] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "NIST special publication 800-90B: recommendation for the entropy sources used for random bit generation," 2018, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf.

[24] J. Kelsey, K. A. McKay, and M. S. Turan, "Predictive models for min-entropy estimation," in *Proceedings of the 17th International WorkshopCryptographic Hardware and Embedded Systems—CHES 2015*, pp. 373–392, Saint-Malo, France, September 2015.

[25] S. Aras and İ. D. Kocakoç, "A new model selection strategy in time series forecasting with artificial neural networks: IHTS," *Neurocomputing*, vol. 174, pp. 974–987, 2016.

[26] J. P. Donate, X. Li, G. G. Sánchez, and A. S. de Miguel, "Time series forecasting by evolving artificial neural networks with genetic algorithms, differential evolution and estimation of distribution algorithm," *Neural Computing and Applications*, vol. 22, no. 1, pp. 11–20, 2013.

[27] J. C. Luna-Sanchez, E. Gómez-Ramírez, K. Najim, and E. Ikonen, "Forecasting time series with a logarithmic model for the polynomial artificial neural networks," in *Proceedings of the 2011 International Joint Conference on Neural Networks, IJCNN 2011*, pp. 2725–2732, San Jose, CA, USA, 2011.

[28] C. de Groot and D. Würtz, "Analysis of univariate time series with connectionist nets: a case study of two classical examples," *Neurocomputing*, vol. 3, no. 4, pp. 177–192, 1991.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016.

[30] X. Cai, N. Zhang, G. K. Venayagamoorthy, and D. C. Wunsch II, "Time series prediction with recurrent neural networks trained by a hybrid PSO-EA algorithm," *Neurocomputing*, vol. 70, no. 13–15, pp. 2342–2353, 2007.

[31] A. Jain and A. M. Kumar, "Hybrid neural network models for hydrologic time series forecasting," *Applied Soft Computing*, vol. 7, no. 2, pp. 585–592, 2007.

[32] J. M. P. Menezes Jr. and G. A. Barreto, "Long-term time series prediction with the NARX network: an empirical evaluation," *Neurocomputing*, vol. 71, no. 16–18, pp. 3335–3343, 2008.

[33] P. Hagerty and T. Draper, "Entropy bounds and statistical tests," 2012, https://csrc.nist.gov/csrc/media/events/random-bit-generation-workshop-2012/documents/hagerty_entropy_paper.pdf.

[34] S. Zhu, Y. Ma, T. Chen, J. Lin, and J. Jing, "Analysis and improvement of entropy estimators in NIST SP 800-90b for non-IID entropy sources," *IACR Transactions on Symmetric Cryptology*, no. 3, pp. 151–168, 2017.

[35] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, USA, 1996.

[36] J. Yang, S. Zhu, T. Chen, Y. Ma, N. Lv, and J. Lin, "Neural network based min-entropy estimation for random number generators," in *Proceedings of the 14th International Conference Security and Privacy in Communication Networks—SecureComm*, pp. 231–250, Singapore, August 2018.