Journal of Informatics
Electrical and Electronics Engineering, 2021,
Vol. 02, Iss. 02, S. No. 022, pp. 1-9
ISSN (Online): 2582-7006

LFSR Next Bit Prediction through Deep Learning

Sanchit Gupta[1], Priyansh Singh[2], Noopur Shrotriya[3], Tanvi Baweja[4]

[1,3]Scientific Analysis Group, Defence Research and Development Organization, Delhi
[2]ABES Engineering College, Ghaziabad, UP
[4]Guru Gobind Singh Indraprastha University, Delhi,
[1]sanchitgupta@sag.drdo.in, [2]priyanshsingh@outlook.com*, [3]noopurshrotriya@sag.drdo.in, [4]tanvibaweja@gmail.com

How to cite this paper: S. Gupta, P. Singh, N. Shrotriya, T. Baweja (2021) LFSR Next Bit Prediction through Deep Learning. *Journal of Informatics Electrical and Electronics Engineering*, Vol. 02, Iss. 02, S. No. 022, pp. 1-9, 2021.

https://doi.org/10.54060/JIEEE/002.02.022

Received: 02/04/2021
Accepted: 23/05/2021
Published: 04/06/2021

Abstract

Pseudorandom bit sequences are generated using deterministic algorithms to simulate truly random sequences. Many cryptographic algorithms use pseudorandom sequences, and the randomness of these sequences greatly impacts the robustness of these algorithms. Important crypto primitive Linear Feedback Shift Register (LFSR) and its combinations have long been used in stream ciphers for the generation of pseudorandom bit sequences. The sequences generated by LFSR can be predicted using the traditional Berlekamp Massey Algorithm, which solves LFSR in 2×n number of bits, where n is the degree of LFSR. Many different techniques based on ML classifiers have been successful at predicting the next bit of the sequences generated by LFSR. However, the main limitation in the existing approaches is that they require a large number (as compared to the degree of LFSR) of bits to solve the LFSR. In this paper, we have proposed a novel Pattern Duplication technique that exponentially reduces the input bits requirement for training the ML Model. This Pattern Duplication technique generates new samples from the available data using two properties of the XOR function used in LFSRs. We have used the Deep Neural Networks (DNN) as the next bit predictor of the sequences generated by LFSR along with the Pattern Duplication technique. Due to the Pattern Duplication technique, we need a very small number of input patterns for DNN. Moreover, in some cases, the DNN model managed to predict LFSRs in less than 2n bits as compared to the Berlekamp Massey Algorithm. However, this technique was not successful in cases where LFSRs have primitive polynomials with a higher number of tap points.
.

Keywords

Linear Feedback Shift Register (LFSR); Multilayer Perceptron (MLP); Deep Neural Networks (DNN); Next bit prediction.

1. Introduction

We use many randomized algorithms which require pseudorandom data for their working. Pseudorandom sequences are not truly random and can be regenerated again with a seed. Cryptographic algorithms use these sequences for their working by assuming their unpredictable property. Robustness of such cryptographic algorithms is highly dependent on the unpredictable property of sequence. That is if an attacker don't have the access of initial seed, he can't predict the future bits of sequence. With the tremendous growth in the area of Machine Learning many attempts have been made to predict such sequences.
Hernández et al. [2000, 2001] converted the prediction problem to classification problem and used decision tree classifier C4.5 to predict pseudorandom sequences generated from LFSR. Their objective was to use ML based next bit prediction models as evaluation criteria. Khan [2005] extended this work and applied it on various LFSRs and tabulated the minimum number of bits required for predicting LFSR generated bits. Kant and Khan [2006] uses this model on other pseudo random generators like Geffe and proposed a True Next Prediction model. A more complex analysis was made by Kant [2009], reporting results for prediction of sequences generated by keystream generators of eSTREAM project. Though they have not been successful to predict eSTREAM candidates, they have reported good results on LFSRs, Geffe Generator and Alternative step generators.
Then use of four machine learning algorithms was made; namely, decision tree based C4.5 algorithm, Naive Bayes, Averaged One Dependence Estimators (AODE) and Multi-Layer Perceptron for the next bit prediction problem. Peinando and Ortiz (2014) used backpropagation Multilayer Perceptron (MLP) to predict sequences generated by LFSR.
In this paper, we apply a different approach - the deep neural networks - to predict the sequences generated by LFSR. We have used Pattern Duplication Technique along with Deep Neural Networks (DNN) for this task. We have used Feedforward Neural Network with two hidden layers in DNN and tested the sequence generated by LFSRs having primitive polynomials of degree 6to 100. With thorough experimentation we have estimated the minimum bound on the number of input bits required by model for each LFSR and tabulated the results. We have found that the results of our model are comparable with the Berlekamp Massey Algorithm.
The results show that the minimum number of bits required by the DNN is at least 7% lesser than the number of bits required by BM Algorithm and this percentage increases with the increase of degree of the LFSR and reaches up to 45% for LFSR of degree 100. The results confirm that these networks can predict the entire sequence knowing fewer input patterns than the existing reported techniques based on other classifiers.

2. Preliminaries

2.1. LFSR

A Linear Feedback Shift Register is a shift register used to generate long pseudorandom sequences. An LFSR can be of any desired length but it has to be given a certain initial state (usually random distribution) and has to be tapped at certain points to feedback one bit after performing a certain linear operation (usually XOR) on the tapped bit. The tapped bits/ tap points are generally decided using primitive polynomials. An LFSR of length m can be constructed using primitive polynomials of degree m, where the period of the LFSR will be $2^m - 1$ bits long. A primitive polynomial is one which is not reducible and has a non-zero constant term. In the field of cryptography, they are used for pseudorandom generators (PRGs). Primitive polynomials are used for PRGs because they can generate the full period of the pseudorandom bits. In general, a primitive polynomial of degree m over a field with two elements will generate $2^m - 1$ pseudorandom bits before repeating the same sequence. An LFSR can be better understood using an example.

For example, an LFSR of degree m = 5. For every LFSR of degree m there exist certain primitive polynomials of degree m, out of which we can choose the one we desire according to the tap points. The primitive polynomials of degree 5 are $x^5 + x^2 + 1$, $x^5 + x^4 + x^2 + 1$ & $x^5 + x^4 + x^3 + 1$. The indices of the polynomials tell us which bits in the LFSR are to be tapped to generate a pseudorandom sequence. These primitive polynomials will generate pseudorandom sequences where the maximum length of the sequence will be 31 ($2^5 - 1$) bits. Figure 1 shows the structure of LFSR using $x^5 + x^2 + 1$ as the primitive polynomial for our LFSR.

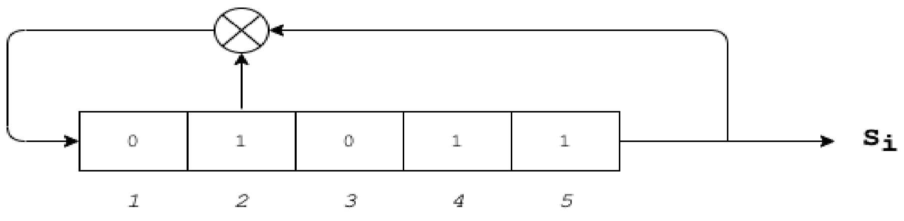


Figure 1. LFSR for $x^5 + x^2 + 1$

2.2. Deep Learning

Deep Learning is a sub-field of machine learning and its algorithms try to mimic the function of neurons in the brain. These algorithms are scalable with the data and generally their performance keeps getting better with the input of more data. Feed-forward neural networks with multiple layers are generally being used in DNN. They have many configurable hyperparameters like activation function, loss function, learning rate etc. which greatly impact the performance of the model. Also the choice of number of layers and number of neurons in each layer are very important decisions which a developer has to take considering the size and type of problem and resource availability.

2.3. Next Bit Prediction Problem

Suppose an adversary is observing a binary sequence b1,b2...bn generated by a deterministic Pseudo Random Bit Generator. Then predicting the next bit i.e. bn+1 bit with the limited information from the past, should not be possible with better than chance probability.

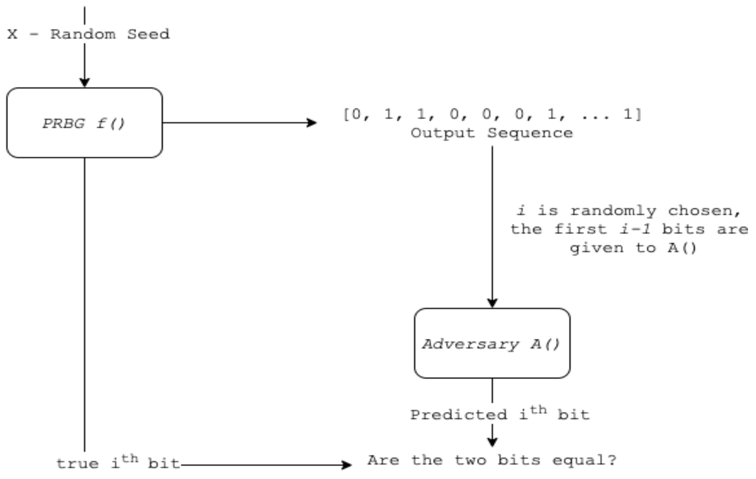


Figure 2. Next Bit Prediction Problem

3. METHODOLOGY

3.1. Pattern Duplication Technique

Hernández et al. [2000, 2001] stated a way to convert the Next Bit Prediction problem into a classificatory problem. In this approach, bits generated by PRBG are converted into patterns of length 'b' (b<<n) in an overlapping way, where b is called frame length. For example, the sequence b1, b2, b3, b4, b5, b6 will be converted into following patterns if frame length is 3:

P1: b1, b2, b3, b4
P2: b2, b3, b4, b5
P3: b3, b4, b5, b6

In each pattern, the last bit is considered as Class Bit and other bits are considered as attributes and these patterns are fed to various classification algorithms. All the existing Machine Learning based research is based on this pattern formulation technique. With this formulation only a limited number of patterns (n- b) are generated. We have developed a Pattern Duplication technique that exponentially increases the number of these patterns. In this technique, a bitstream is generated using an LFSR and then, initial patterns are formed using the same way as suggested by Hernández et al. [2000, 2001]. Then different combinations of these initial patterns are formed and for each combination, the shift and add and the inversion property of sequence operations are used to create new patterns. The *'Shift and Add'* property of sequences generated by the XOR operation states: the modulo-two addition of any two identical pseudorandom sequences with different phases generates another identical pseudorandom sequence but with another phase. The pseudocode of the module for implementing *'Shift and Add'* property is given in Module 1.

Module 1: Module for Shift and Add property

```
Function shift_and_add
    Pass In: combination of patterns c
    foo ← list of 0s equal to length(pattern)
    for every pattern p in c:
        for i from 0 to length(p)-1:
            foo[i] ← foo[i] xor p[i]
        endfor
    endfor
    append foo to new_patterns
    Pass Out: new_patterns
Endfunction
```

The *'Inversion'* property of sequences generated by the XOR operation states: if all the attribute bits of patterns are inverted, the class bit remains the same with the assumption that size of subsequence should be equal to the degree of LFSR. The pseudocode of the module for implementing 'Inversion' property is given in Module 2.

Module 2: Module for Inversion property

```
Function Inversion
    Pass In: single pattern
    for i from 0 to length(pattern)-2:
        invert pattern[i]
    endfor
    append pattern to new_patterns
    Pass Out: nothing
Endfunction
```

Using these two properties, a new pattern can be formed with two or more existing patterns. Furthermore, these new patterns can be used to generate more patterns. The process of pattern creation is repeated till a sufficient number of unique patterns are created. The number of patterns generated depends upon many factors like the storage capacity of the computational system, maximum possible patterns and need for retraining the model. In our case, we have generated around 15000 patterns (or maximum possible patterns: $2^{\text{framelength}}$). The pseudocode of the module for implementing 'Pattern Duplication' property is given in Module 3.

Module 3: Module for Pattern Duplication

```
Function Pattern_duplication
    Pass In: dataset, max_counter
    while (count(unique values in new_patterns) $\leqslant$ max_counter)
        combinations $\leftarrow$ nested list of all possible combinations of patterns in dataset taken 2,3,4...length(dataset)-1
        at a time
        for each combination c in combinations: inversion (Shift_and_add (c))
    end while
    Pass Out: new_patterns
Endfunction
```

Valid patterns are formed only when the frame length is equal to the degree of the LFSR. Therefore, an initial frame length is chosen and incremented iteratively till good classification results are obtained. In the next step, the number of initial bits generated by the LFSR is repeatedly reduced to get a lower bound on the number of bits required to successfully predict the LFSR.

3.2. Prediction using Deep Learning

Initially, some LFSRs of different configurations (degree, tap points, primitive polynomials) were taken and sufficient patterns were generated using the Pattern Duplication technique. Numerous tests were performed with the DNN program various combinations of hyperparameters like number of layers, number of nodes in each layer, activation functions, learning rates etc. The goal here was to find which parameters would result in a lesser value of loss function. A good model here is defined as one in which the training accuracy tends to meet the validation accuracy.

Based upon extensive experimentation using KERAS library we were able to freeze following hyper parameters for the deep learning model:

1. Number of Internal Layers: 2 (with number of nodes = frame length)
2. Regularization : L2
3. Activation Function: Relu (internal layers) & Sigmoid (final layer)

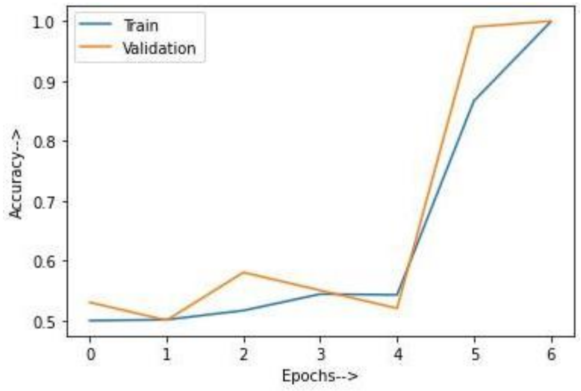
4. Optimization Function: Adam
5. Learning rate : 0.001
6. Loss function : Binary Cross Entropy

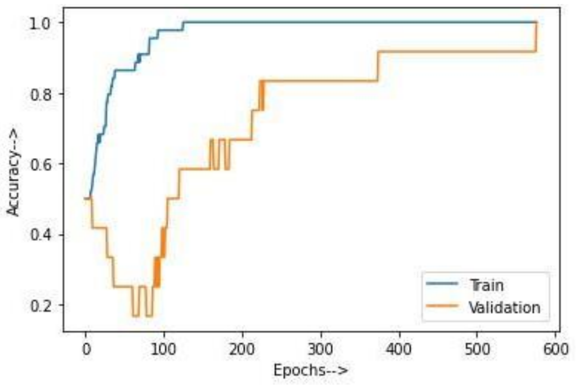
After freezing the best hyper parameters, the model is tested with different LFSRs and estimation of the minimum amount of bits required for successful prediction is done. After reaching a certain threshold where the validation accuracy was close to 100% consistently, the program was terminated and the results were recorded

4. OBSERVATIONS AND RESULTS

After applying the Pattern Duplication technique, the final dataset is split into training, testing and validation sets. Training accuracy is calculated as how often the predicted values match the true values present in the training set. Validation dataset has at least 100 patterns and model performance is evaluated using this dataset after each epoch. This accuracy is called validation accuracy. The plot of training and validation accuracy against the number of epochs for some polynomials is given below. For each LFSR polynomial, the graph on the left is plotted once training and validation accuracy reaches 1.0 (100%) and testing accuracy is recorded using the testing set, which has around 100 patterns. Frame length is recorded at this instance and then the model is retrained by fixing this frame length with repeatedly reducing the number of bits generated by the LFSR so that we achieve around 100% validation and testing accuracy. The graph on the right is plotted on reaching the lower bound on the number of bits. This number of bits is also recorded. Finally, the Model is tested with the test data set for final conformance. The results confirm that around 100% validation and testing accuracy is achieved only when frame length is equal to the degree of the primitive polynomial (or the degree of the LFSR).
However, similar results weren't obtained for some primitive polynomials with seven terms. For example, while working with LFSR of degree 54, 100% training and validation accuracy couldn't be achieved using the primitive polynomial $x^{54} + x^6 + x^5 + x^4 + x^3 + x^2 + 1$ at any frame length up to 54. But once a primitive polynomial with five terms , i.e., $x^{54} + x^{53} + x^{49} + x^{39} + 1$, was used for the same LFSR, 100% training and validation accuracy was achieved at frame length of 54. After fixing this frame length and repeatedly reducing the number of bits generated by the LFSR, the minimum bound was found at 58 bits with 100% training, validation and testing accuracy. Table 1 shows some of our observations in tabular form. In all the given cases we were able to solve LFSR in less than 2×n bits.

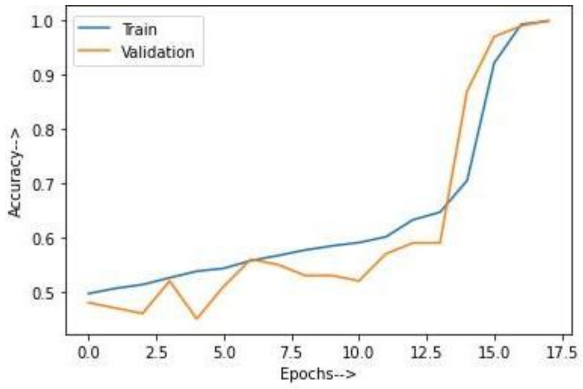
$x^{23} + x^5 + x^4 + x + 1$

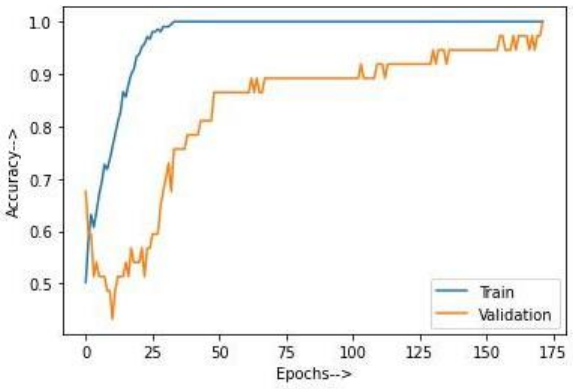




Frame Length:	23	No. of bits:	59
Training Accuracy:	1.000	Training Accuracy:	1.000
Testing Accuracy:	1.000	Testing Accuracy:	1.000

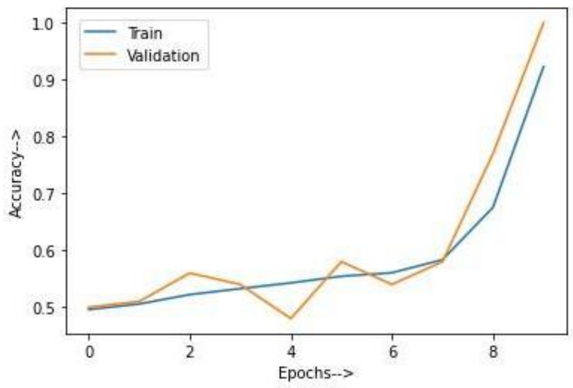
$x^{50} + x^4 + x^3 + x^2 + 1$

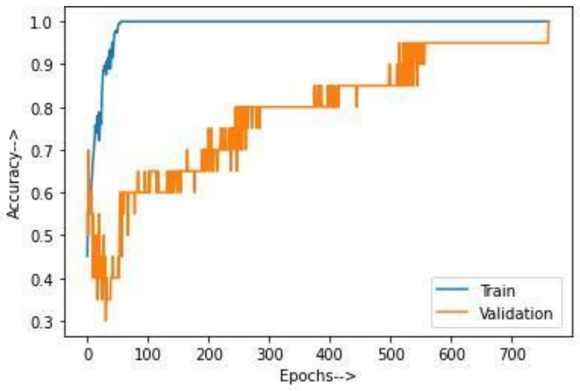




Frame Length:	50	No. of bits:	58
Training Accuracy:	1.000	Training Accuracy:	1.000
Testing Accuracy:	1.000	Testing Accuracy:	1.000

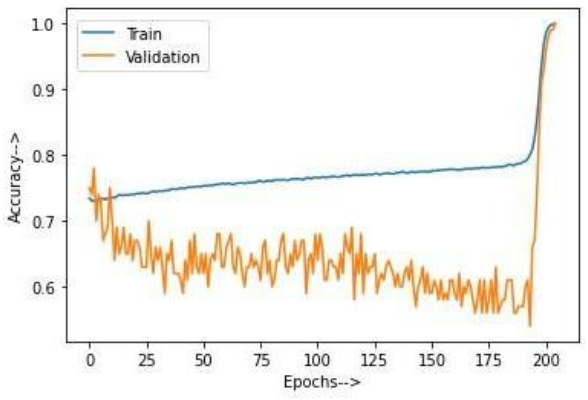
$x^{54} + x^{53} + x^{49} + x^{39} + 1$

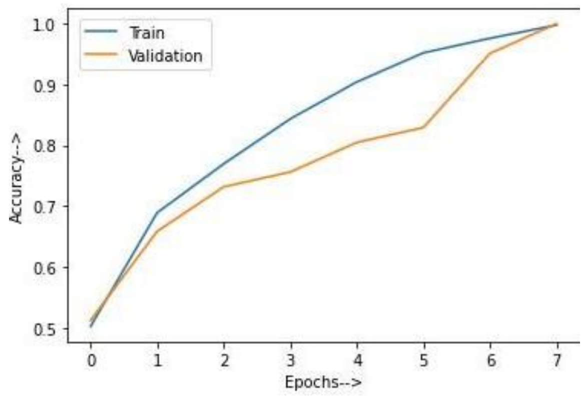




Frame Length:	54	No. of bits:	58
Training Accuracy:	0.994	Training Accuracy:	1.000
Testing Accuracy:	0.998	Testing Accuracy:	1.000

$x^{83} + x^7 + x^4 + x^2 + 1$

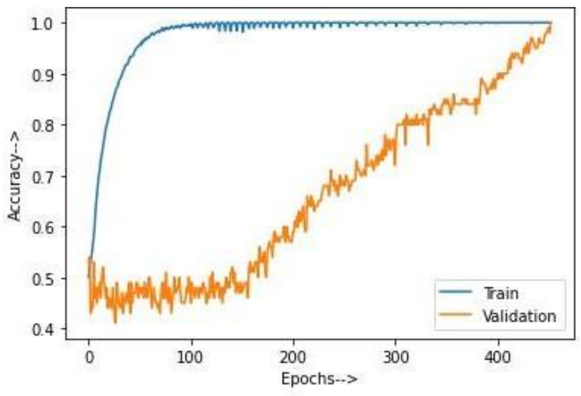


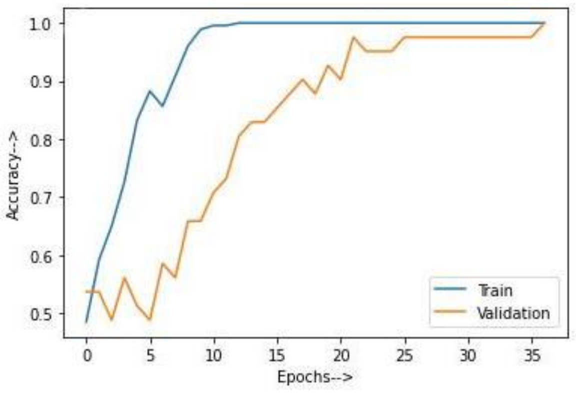


Frame Length:	83
Training Accuracy:	1.000
Testing Accuracy:	0.998

No. of bits:	92
Training Accuracy:	1.000
Testing Accuracy:	1.000

$x^{100} + x^8 + x^7 + x^2 + 1$

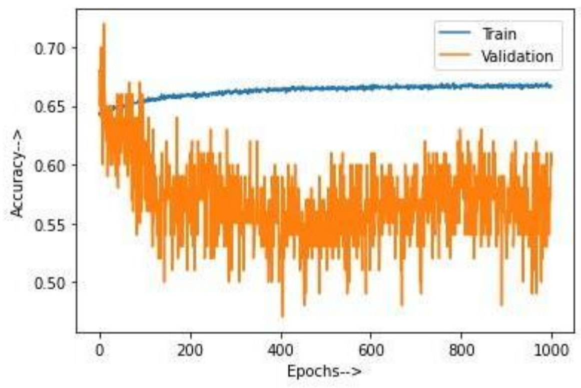




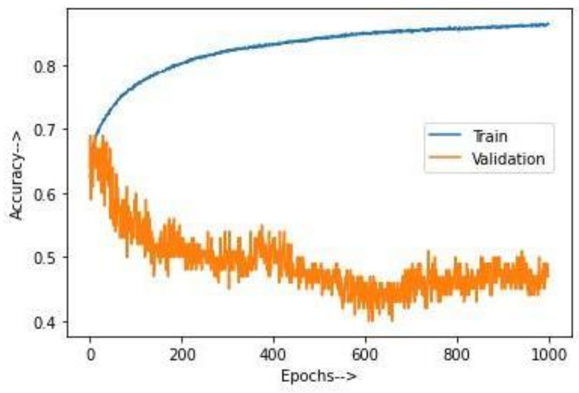
Frame Length:	100
Training Accuracy:	1.000
Testing Accuracy:	1.000

No. of bits:	109
Training Accuracy:	1.000
Testing Accuracy:	1.000

$x^{54} + x^6 + x^5 + x^4 + x^3 + x^2 + 1$



$x^{72} + x^6 + x^4 + x^3 + x^2 + x^1 + 1$



Frame Length:	54 (FAILED)
Training Accuracy:	0.909
Testing Accuracy:	0.835

Frame Length:	72 (FAILED)
Training Accuracy:	0.921
Testing Accuracy:	0.586

Table 1. LFSR Prediction results

Degree of Polynomial	Polynomial	Frame Length	Minimum no. of Bits	No. of bits required by BM Algorithm
15	$x^{15} + x^{10} + x^5 + x + 1$	15	**22**	30
23	$x^{23} + x^5 + x^4 + x + 1$	23	**29**	46
38	$x^{38} + x^6 + x^5 + x + 1$	38	**44**	76
50	$x^{50} + x^4 + x^3 + x^2 + 1$	50	**58**	100
54	$x^{54} + x^{53} + x^{49} + x^{39} + 1$	54	**62**	108
54	$x^{54} + x^6 + x^5 + x^4 + x^3 + x^2 + 1$	-	**-**	-
72	$x^{72} + x^{23} + x^{14} + x^2 + 1$	72	**81**	144
72	$x^{72} + x^6 + x^4 + x^3 + x^2 + x^1 + 1$	-	**-**	-
75	$x^{75} + x^6 + x^3 + x^1 + 1$	75	**84**	150
83	$x^{83} + x^7 + x^4 + x^2 + 1$	83	**92**	166
100	$x^{100} + x^8 + x^7 + x^2 + 1$	100	**109**	200

5. CONCLUSION

The use of DNN and the Pattern Duplication technique to solve the next bit prediction problem for LFSR of degrees up to 100 has proven successful. The results confirm that the Pattern Duplication technique has been successful in generating new patterns using the *'Shift and Add'* and the *'Inversion'* property. The minimum number of bits required by the DNN to solve LFSR is significantly less than what is required by the Berlekamp-Massey Algorithm. The proposed methodology did not work for some LFSRs having primitive polynomial of a certain degree with seven terms. However, desired results were obtained for primitive polynomial of the same degree having three or five terms.

References

[1]. A. J. Menezes, P. C. V. Oorschot, & S. A. Vanstone, "Handbook of applied cryptographic Press, 2018.

[2]. J.C. Hernandez, J. M. Sierra, C. M. Perera, et al.," Using the General Next Bit Predictor Like an Evaluation Criteria," proceedings of NESSIE workshop Leuven, Belgium,2000.

[3]. S.S. Khan, "Classificatory Prediction and Primitive Polynomial Construction of Linear Feedback Shift Registers using Decision Tree Approach." KBCS-fifth International Conference cn Knowledge Based Computer Systems. 2004.

[4]. S. Kant and S. S. Khan, "Analyzing a Class of Pseudo-random Bit Generator through Inductive Machine Learning Paradigm", Int. J. Intelligent Data Analysis, vol. 10, no.6, pp. 539-554,2006.

[5]. S. Kant, N. Kumar, S. Gupta, et al., "Impact of machine learning algorithms on analysis of stream ciphers," *Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*, pp. 251-258,2009,

[6]. T.Mitchell, "Machine Learning," McGraw Hill,vol18,no.3,1997.

[7]. E. Casey," Bredekamp-Massey Algorithm.," REU Summer Report, University of Minnesota, Minneapolis, MN,pp.1-10,2000.

[8]. W. Stahnke, "Primitive binary polynomials," Math. Comput., vol. 27, no. 124, pp. 977–980, 1973.

[9]. Y. Lin, "'Shift and add' property of m-sequences and its application to channel characterisation of digital magnetic recording," IEE Proc. - Commun., vol. 142, no. 3, p. 135, 1995.