Spring 2023

Predicting Jamming Systems Frequency Hopping Sequences Using Artificial Neural Networks

Charles Strickland

PREDICTING JAMMING SYSTEMS FREQUENCY HOPPING SEQUENCES USING ARTIFICIAL NEURAL NETWORKS

by

CHARLES STRICKLAND

(Under the Direction of Rami J. Haddad)

ABSTRACT

This work proposes a neural network architecture that was designed to predict and reverse engineer frequency hopping jamming systems. The neural network was initially optimized for use with a 12th order linear shift feedback register maximum length sequence utilizing a minimal polynomial as the characteristic polynomial. This neural network was then scaled to accommodate 7 different sequences, of orders 6 through 12. The neural network was trained for these sequences using training data that is 10 times the length of the sequence. This information is then used to generate a hopping sequence that reduces the jamming interference to 0 with as few as 4 jammer hopping samples. The model is also capable of determining if the jammer is utilizing a sequence that the model is trained for in as few as 25 jammer hopping samples.

INDEX WORDS: Frequency Hopping, Artificial Intelligence, Jamming, Neural Network, Random Number Generators, Pseudo Noise, Maximum Length Sequences

PREDICTING JAMMING SYSTEMS FREQUENCY HOPPING SEQUENCES USING ARTIFICIAL NEURAL NETWORKS

by

CHARLES STRICKLAND

B.S., Georgia Southern University, 2017

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

PREDICTING JAMMING SYSTEMS FREQUENCY HOPPING SEQUENCES USING ARTIFICIAL NEURAL NETWORKS

by

CHARLES STRICKLAND

Major Professor:	Rami J. Haddad
Committee:	Sungkyun Lim
	Seungmo Kim

Electronic Version Approved:
May 2023

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Noise jamming systems can be harmful to the signal integrity of a radio-frequency (RF) system, reducing and even eliminating the ability of a transmitter to send a signal or a receiver to receive a signal. One method used to defend against these attacks is the use of frequency hopping. Frequency hopping schemes help to alleviate these effects by constantly moving the transmitted signal to different frequencies, thus reducing the probability that a jamming system can effectively inhibit communication entirely. However, frequency hopping does not entirely eliminate the jamming effect. As demonstrated in [1], it can be difficult to eliminate the effects of a noise jamming attack by just frequency hopping. They show that when in the presence of a high-power noise jammer it can be difficult to remove the jamming effects from the transmitted signal.

Frequency hopping jammers, also known as noise jammers, jam signals by emitting RF energy while hopping over a specified frequency range [2]. These frequencies and the order that they are hopped to, known as the hopping sequence, are typically chosen by using a pseudo-random number generator (PRNG). For this research, a linear-feedback shift register (LSFR) that generates maximum-length sequences (m-sequences) was chosen as the PN PRNG. M-sequences are simple to implement and do not take up much memory since they use LSFRs for the sequence generation. For these reasons, they are commonly found in frequency hopping systems. The primitive polynomials chosen for the m-sequences are minimal polynomials which are the most common characteristic polynomials used for these sequences since they minimize the memory, hardware, and computations required. Due to the highly nonlinear properties of PRNGs, a neural network was chosen as the machine learning application.

Previous research has been conducted into detecting and classifying different types of jamming signals present in a given spectrum, as demonstrated in [3] and [4]. The research presented in this paper is intended to build upon this previous work and allow for frequency hopping systems to implement moving target defense. Once a frequency hopping jammer has been detected and classified, the model proposed in this paper can be utilized to predict where the jammer is going to hop after sampling a sequence of 4 consecutive hops. This information can then be used in smart frequency hopping systems to hop around the interfering signal transmitted by the jammer.

The artificial neural network presented herein has the capacity to learn multiple pseudo-noise (PN) sequences and can predict the next value in the sequence with a high degree of accuracy. This network consists of an input layer with 4 neurons, 4 hidden layers, each containing 1024 neurons, and an output layer with a single output neuron. This network was trained for 7 different maximum-length sequences (m-sequences), whose primitive polynomials were orders 6 through 12. The primitive polynomial for each sequence was chosen as a minimal polynomial. An algorithm was also developed that allows for the system to determine if the jamming system is using a primitive polynomial that is known by this system.

The remainder of this work is structured as follows. Chapter 2 contains background information and the literature review, which discusses existing work. Chapter 3 describes the details of the simulation design and the parameters used. Chapter 4 contains the results and provides some discussion. Chapter 5 concludes this work and suggests direction for future work.

CHAPTER 2
LITERATURE REVIEW

2.1 Frequency Jamming

Due to the increased reliance on wireless networks, transmission security has become an increasingly important and necessary area of study. These wireless networks are especially vulnerable to jamming attacks due to how simple and easy the attacks can be to implement [5].

Frequency jamming occurs when electromagnetic energy is purposefully emitted in the direction of a communication system with the intent of interfering with the signal transmission of the system. This is accomplished by transmitting signals at the same frequency as the targeted communication [6]. This technique is often used for military and defense purposes, as it can block enemy communication and disrupt their operations. There are two main types of frequency jamming: reactive and cognitive.

2.1.1 Reactive Jamming

Reactive jamming involves a jamming system that listens to the RF spectrum and is able to sense when a radio transmission is being sent [6, 7]. It will then continuously transmit a strong signal on the same frequency as the target communication, effectively drowning it out and preventing it from being received. This can be further broken down into noise-spoofing and noise-jamming [7]. Noise-spoofing is when the jammer targets a specific transmitter and wants to prevent it from being able to send a signal. This is accomplished by transmitting a strong signal over the transmitter's channel so that the transmitter thinks that the channel is busy. While Noise-jamming is when the jammer targets one or more receivers and has the goal of preventing them from receiving a complete packet. This is accomplished by sending a signal that is stronger than the one sent by the transmitter so that the receivers are not able to receive the signal accurately [7].

Based on the above, reactive jamming can be effective in certain situations, such as when the target signal is relatively weak or when the jammer is located relatively close to the target. It can also reduce the packet delivery ratio to nearly 0 [6]. However, it requires a strong and continuous signal, which can be difficult to sustain for long periods of time. Additionally, reactive jamming can be relatively easy to detect, as it involves transmitting a strong signal at the same frequency as the target. Reactive jamming is often used for military and defense purposes, as it can be used to block enemy communication and disrupt their operations. It can also be used in other fields where secure communication is important, such as in emergency response or transportation.

2.1.2 Cognitive Jamming

On the other hand, more complex jamming systems have been developed that can adapt their output signal to various situations. These cognitive jammers sense the spectrum in real-time, quickly analyze and characterize the data, and develop and deploy the jamming strategy [7]. This effectively transmits a signal similar to the target communication but with added noise or false information. Noise jamming transmits a broad spectrum of noise over a wide range of frequencies. This can confuse the receiver and make it difficult for them to interpret the message. Each type of jamming has its advantages and disadvantages, depending on the specific situation. Cognitive jamming can be more subtle and difficult to detect, but it is more complex to execute and less effective against advanced communication systems.

2.2 Frequency Hopping Jamming Systems

Frequency hopping jamming systems, also known as noise jammers, are a type of cognitive jammer that try to prevent communication by mimicking noise and overloading the

RF spectrum with energy to make communication difficult. They are capable of jamming multiple channels simultaneously [2]. They hop over a given spectrum with sufficient power so that the receiving system can not distinguish between the ”noise” and the signal of interest. This jamming waveform is generated by using a frequency hopping sequence that is developed by using a pseudo-random number generator. This helps the jamming system to appear as noise in the background of the spectrum that the RF system is trying to communicate over [8].

2.2.1 Frequency Jamming Countermeasures

Countermeasures to frequency jamming include using encryption to secure communication, switching to alternative communication methods or frequencies, changing the transmission rate, using ambient backscatter communication, utilizing dynamic power control, using modern machine learning methods, and using techniques such as spread spectrum to make the target communication more difficult to jam [9].

Frequency hopping is a proven method to improve the communication reliability of a wireless system. This relatively simple technique is used in Bluetooth communication as a preventative measure to minimize noise and jamming attacks and to increase the integrity of the signal. A study was conducted that showed when WiFi is in the presence of a reactive jamming signal, the transmission accuracy of the signal can be significantly improved [5]. The effectiveness of frequency hopping for jamming protection can be improved even more by increasing the rate that the frequency hopping occurs [10]. Still, frequency hopping can struggle to eliminate the effects of jamming systems that are not stationary, whether that be sweep jamming, frequency hopping jamming, etc [11].

Frequency jamming is a complex and constantly evolving field, with advances in technology and communication systems constantly challenging the effectiveness of jamming techniques. Despite its limitations, frequency jamming remains an important tool in the military and defense sectors, as well as in other fields where secure communication is crucial.

2.3 Detection and Classification of Jamming Presence

Several studies have been conducted to utilizing machine learning algorithms to detect when a jamming signal is present and then to classify what type of jamming signal it is, as shown in [12], [13], [14], [3], and [4]. For these applications, this information can be used to determine whether the system should continue transmitting for stop until the jamming system has ceased. This section will dive deeper into some of the research that has been completed in these areas.

There have been several studies conducted on detecting and classifying the various types of jamming systems. Three options are shown and compared in [3] that utilize machine learning methods (neural networks, random forest, and support vector machines) in order to accurately detect when a jamming signal is present. They were able to achieve 94% accuracy using neural networks and 96% accuracy using random forest methods.

In [4], the authors demonstrate that machine learning algorithms can be used to classify jamming attacks as either Single-tone, barrage, protocol-aware, or successive-pulse. They were able to achieve a 93% success rate.

2.3.1 Frequency Hopping Sequences

When the carrier signal of a transmitted signal changes, or ”hops” over time, this is called frequency hopping. The is usually a set time interval that the signal hops from one frequency to another. The method in which the order of the frequencies is chosen is called the frequency hopping sequence [15].

Frequency hopping is typically used in order to minimize the effect of jamming systems and also to increase the transmission security of the signal, in return increasing the total security of the transmitted signal [5].

There are an infinite number of ways to generate frequency hopping sequences. The only limitation is what the hardware is capable of handling and how fast the hopping needs to occur. For more sensitive information being sent, more complex frequency hopping algorithms will be used. These more complex algorithms require more expensive hardware as more computations will have to be conducted in between each hop. Using something simple like m-sequences can provide fast, cheap, and sufficiently long sequences so that the system can have the security of a frequency hopping system while still keeping expenses and complexity low.

2.4 Maximum-Length Sequences

Maximum-length sequences are used in basic cryptography systems as shown in [16], [17] and [18]. They can also be used in frequency hopping systems to help select the frequency hopping code. These sequences are easy to implement and do not require many hardware resources. They utilize a linear-feedback shift register that has the same number of stages and bits as the order of the chosen primitive polynomial. The use of a primitive polynomial ensures that the sequence generated is of maximum length. The output value of the register is then used for the generation of the sequence [19].

Hopping patterns can then be generated from the resulting maximum-length sequence by combining bits from the sequence based on the number of frequencies, or channels, that are needed [19].

There are certain criteria that m-sequences must meet. The first is that maximum-length sequences should adhere to the ”balanced property’ that states there should be approximately the same number of 1s and 0s in each sequence period. They should also contain consecutive sub-sequences of 1s and 0s within each of the sequences; half of them should have a length of 1, a fourth should have a length of 2, and so on. There should be a periodic autocorrelation function that assumes binary values. When modulo 2 is performed on the sequence with a shifted copy of the sequence, this should result in a different shifted copy of the original sequence [20, 21].

M-sequences use linear-shift feedback registers to generate the sequence. These registers utilize 'taps' that go to an XOR gate at the points where the primitive polynomial used has non-zero coefficients. Because of this, the most common application will use a minimal polynomial for a given primitive polynomial order.

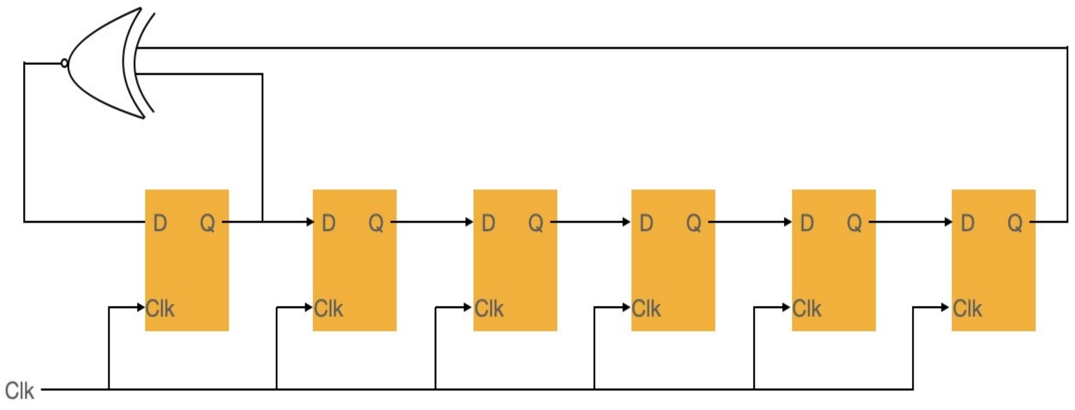


Figure 2.1: Example Linear-Shift Feedback Register

The m-sequences used in this paper were generated using an LSFR model in Matlab. An example LSFR is shown in Figure 3.4. The primitive polynomial chosen for a given m-sequence determines the "taps" where the polynomial has non-zero coefficients. In the example given here, the polynomial used would be $x^7 + x + 1$. Note that the number of flip-flops equals $n - 1$ of a polynomial of degree n.

An LSFR is a method used for generating m-sequences. These generate a pseudo-random sequence that is pseudo-random by nature, which means that the m-sequence is capable of passing the statistical requirements of true random sequences [22]. The LSFR design is often the first choice in many applications requiring a PRNG due to its simplicity and its ability to pass the most basic randomness requirements for various applications [23].

2.4.1 PRIMITIVE POLYNOMIALS

For an m-sequence to achieve the maximum possible length of the sequence, $2^n - 1$ for an n-stage LSFR, the characteristic polynomial of the LSFR must be an irreducible (or primitive) polynomial. A primitive polynomial is defined as a ”monic polynomial that divides $1 + x^k$, for $k = 2^n - 1$.” When a polynomial that meets these criteria is used, the period of the m-sequence will be equal to $2^n - 1$ [24].

[25] describes a 4-part test to determine if a polynomial is primitive or not. The first 2 of these remove all of the reducible polynomials from the list. The final 2 stages finish the test and determine if the given polynomial meets the necessary and sufficient requirements for being considered primitive [25].

In the first stage, the polynomial of degree n that is under test is checked to see if all of the terms are an even power. If this is the case, then the polynomial is rejected as being reducible. If the polynomial passes this test, it moves on to the second stage, where the Euclidean algorithm is used to calculate the greatest common divisor of the polynomial and $x^(2m) + x$ for all m between 1 and $n/2$. This polynomial will fail this second stage if each iteration over m is not equal to 1. Upon passing the second stage, the polynomial moves on to the third stage, where it is verified if the polynomial can divide $x^(2n) + x$. This effectively checks to see if the period of the polynomial is can divide $2^n - 1$. This is necessary to be true in order to check for machine errors that may occur in the second stage. The fourth and final stage checks the modulo function of $x^((2^n - 1)/q)$ with the polynomial for each prime factor q of $2^n - 1$. If any of the results give 1, the polynomial is not primitive [25].

2.4.2 Minimal Polynomials

The more ”taps” that are required to implement a specific LSFR, the more complex the LSFR is, the slower it will perform, and more computations are required between each output value. A weight can be determined for a given polynomial by adding up the total number of ”taps” required to implement the polynomial in the LSFR. Note that m must be an odd number in order to obtain the maximum sequence length possible. The lower the weight, the better the performance of the LSFR since there is less work to be done to generate the sequence. In general, the smallest weight polynomials will have either 3 or 5 terms which equate to weights of 3 and 5, respectively [24]. Some examples of minimal polynomials can be found in 4.5 below.

Table 2.1: Some Example Minimal Polynomials

Polynomial Order	Polynomial
6	$x^6 + x + 1$
7	$x^7 + x + 1$
8	$x^8 + x^7 + x^2 + x + 1$

Minimal Polynomial Uses

- Matlab - In the Matlab function pnsequence(), the default polynomial used is a minimal polynomial [26].

- liquidsdr - In the liquidsdr function msequence(), the default polynomial used in a minimal polynomial [27].

2.4.3 Calculating Primitive Polynomials

Research has also been conducted in [28] where a model is proposed that can accurately calculate the primitive polynomial used in a given m-sequence by using linear algebra. The catch here is that they must sample an entire sequence before they can calculate the polynomial used. This can be an issue for very long sequences and if the seed is changed before an entire sequence has been sent. For the model presented herein, the system only needs to pause for 4 jammer hops before accurately predicting the jammer hopping sequence. This does require a constant seed value for the 4 consecutive hops.

2.4.4 Summary

Using similar methods presented in these papers, these can be built upon to develop a smart frequency hopping system that can detect when a frequency hopping jamming system is present and then update the transmitting frequency hopping sequence so that the signal integrity is preserved and there is minimal interference from the jamming system.

Based on the literature review presented here, it can be reasonably assumed that a frequency-hopping jamming system will most like be utilizing an m-sequence with a minimal polynomial as the characteristic polynomial of the LSFR.

CHAPTER 3

PROPOSED JAMMING PREDICTION NEURAL NETWORK

In order to reverse engineer and predict the hopping sequence of a jamming system, the transmitting system needs to know how many channels the jammer is using, how long the sequence of pseudo-random numbers is, and what type of PRNG the jammer is using. A neural network is presented that can predict with high fidelity where a given jammer is going to hop next over a range of minimal primitive polynomials. For the purposes of this research, certain assumptions had to be made. These assumptions can be found in section 3.1.

Figure 3.1 shows a typical frequency hopping system. In this diagram, a jamming signal is shown, and the frequency hopping sequence generators are the hopping sequences described herein. The change in hopping sequence that is generated at the receiving side of the system is sent to the transmitting system via an out-of-bound (OOB) signal.

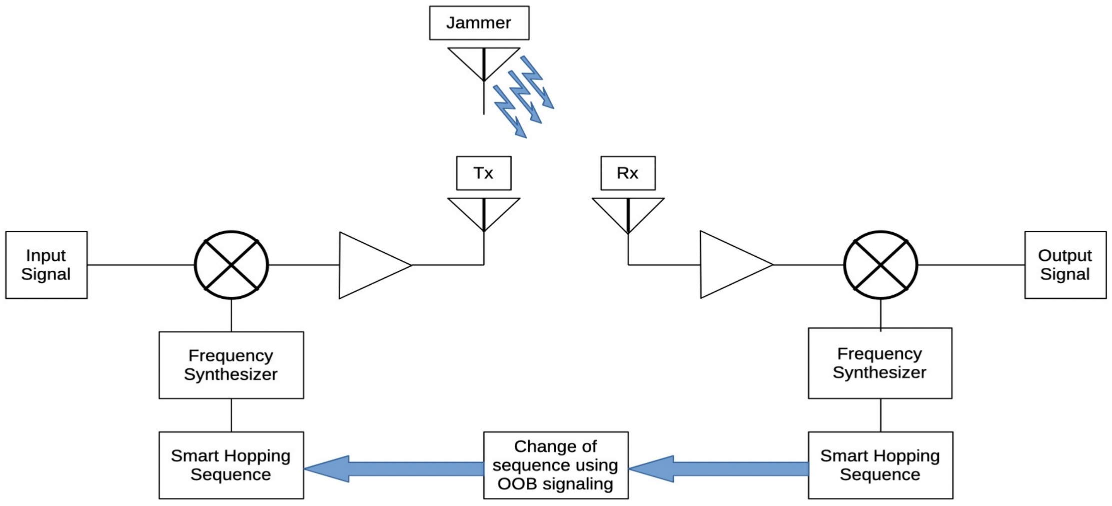


Figure 3.1: Neural Network Architecture for Prediction of a Single Polynomial

3.1 Assumptions

Because of the vast number of possibilities when it comes to methods of determining a frequency hopping algorithm, the following assumptions were made. First, we assume that the jamming system uses PN PRNGs in the generation of the hopping sequence. This is because PN PRNGs are the most common frequency hopping sequence generator and they are simple, fast, and do not require much hardware or memory. Next, The PN PRNG uses minimal polynomials as the characteristic polynomial of the LSFR. These require a minimal number of XOR gates in the feedback, which results in a minimal number of computations for each clock cycle, further simplifying the jamming system. Thirdly, we assume that both the transmitting and jamming systems are both hopping over the same frequency band, and each uses 256 channels in that band. Forth, we assume that both systems are hopping at the same rate. The fifth assumption is that both systems use 8-bit sequences. Sixth is that the jamming system uses an m-sequence that the neural network was trained for. The final assumption is that the transmitting system has already detected the jamming system and has begun to track it.

Based on the literature review presented in 2, the use of PN PRNGS that utilize minimal polynomials are the most likely candidates to be used as the hopping sequence generator in a frequency hopping jamming system since they are easy to implement, take up the least amount of resources, and are fast. Only one minimal polynomial per polynomial degree was used in this research, but it can easily be scaled to accommodate any number of desired polynomials.

3.2 Characteristic Polynomial Selection

The first step in this process was to choose a set of primitive polynomials for 7 different orders. These were chosen as 6th through 12th-order minimal polynomials. These are shown in Table 3.2. These polynomials were chosen to be some of the most likely candidates for the LSFR based on the minimal polynomial characteristics described in this paper.

Table 3.1: Minimal Polynomials Used

Polynomial Order	Polynomial
6	$x^6 + x + 1$
7	$x^7 + x + 1$
8	$x^8 + x^7 + x^2 + x + 1$
9	$x^9 + x^4 + 1$
10	$x^{10} + x^3 + 1$
11	$x^{11} + x^2 + 1$
12	$x^{12} + x^6 + x^4 + x + 1$

After these polynomials were chosen, they were run through the pnsequence() function in Matlab for an entire sequence and used to generate training data for training the neural network.

3.3 Neural Network Architecture

To start, a neural network architecture was developed, designed, and trained to predict only the 12th-order primitive polynomial. This was done as a proof of concept to determine the complexity of the neural network required in order to accurately predict these hopping sequences.

An optimization effort was taken in order to ensure accurate results and that the chosen architecture is capable of learning the hopping sequences. This optimization effort involved varying the number of inputs, the number of hidden layers, the widths of each hidden layer, and the number of training sets sent through the network during the training process.

Ultimately, the neural network chosen has 4 inputs, 5 fully-connected hidden layers, each with 256 neurons, and a single output. The 4 inputs are any 4 sequential numbers from the 8-bit m-sequence. Once the neural network was optimized to predict the 12th-order polynomial, the same architecture was used to train for the remaining polynomials and achieved similar results.

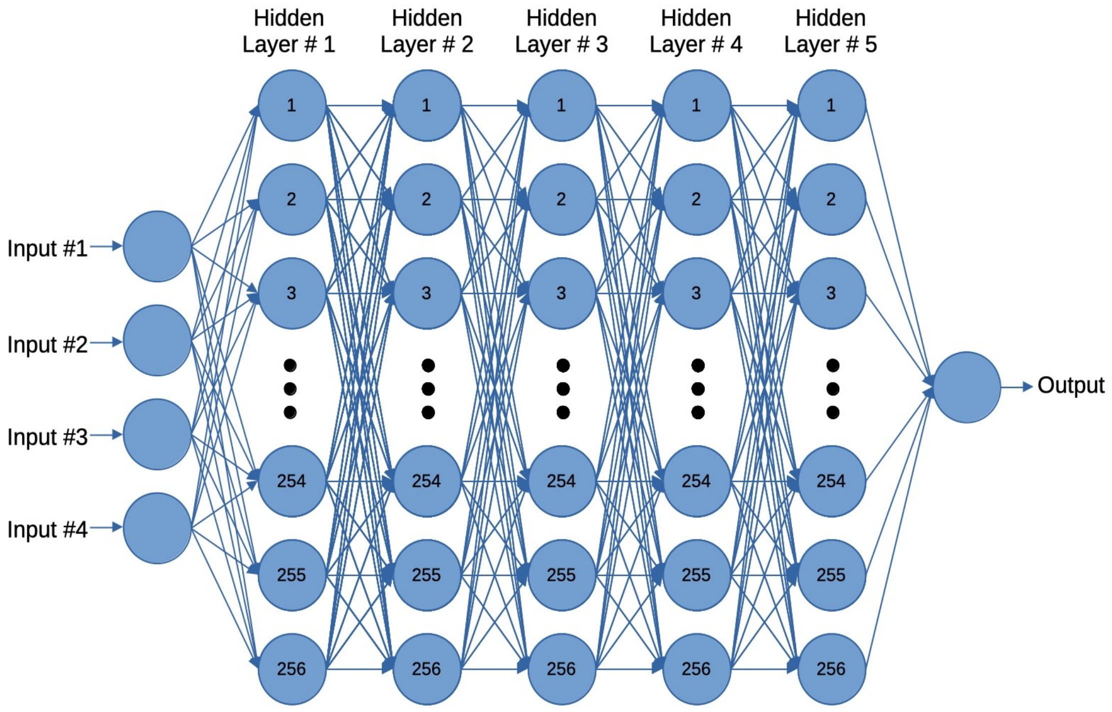


Figure 3.2: Neural Network Architecture for Prediction of a Single Polynomial

Finally, a larger neural network was designed and trained that was able to accurately predict the pseudo-random sequence of all seven of the chosen primitive polynomials, regardless of the seed value, as long as there were 4 consecutive samples of the sequence data. Based on the optimization effort, it was shown that increasing the width of the neural network improves performance much better than increasing the depth of the network. For this case, the original network was used with the width of all of the hidden layers updated to 1024 neurons.

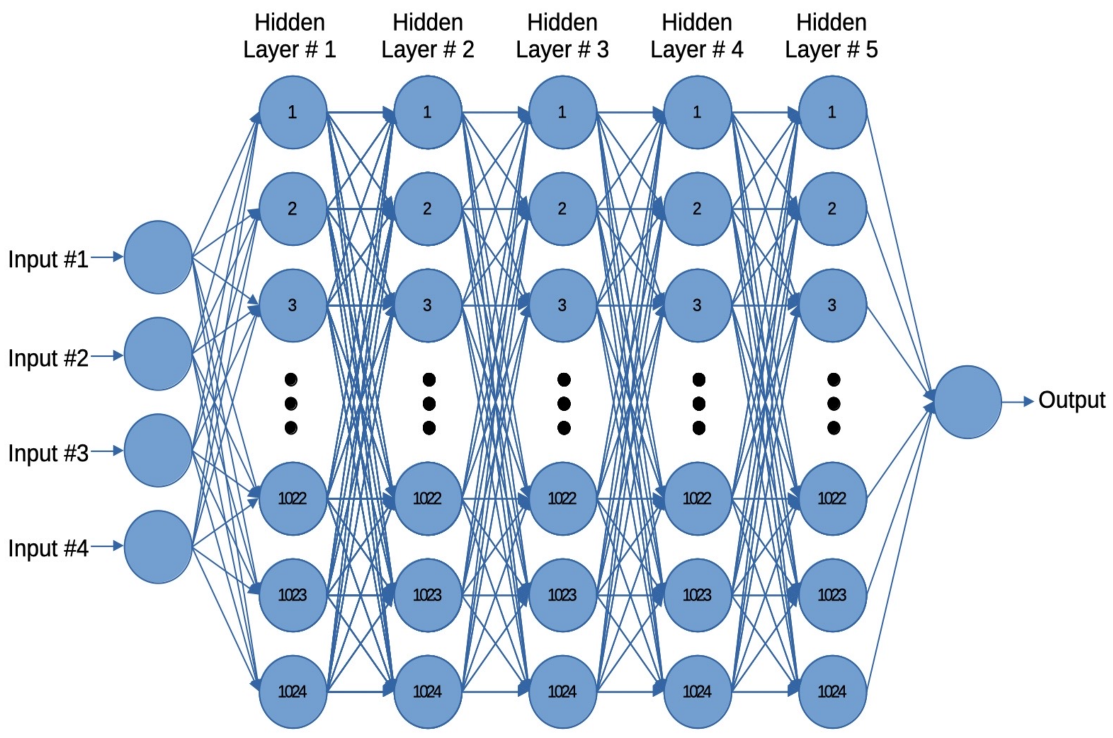


Figure 3.3: Neural Network Architecture for Prediction of All Polynomials

3.4 Neural Network Training

The training data used for the individual neural networks consisted of 10 times the given sequence length for the targeted polynomial. Since the 6th-order polynomial has a sequence length of 63 ($2^6 - 1$), then the training set consisted of 630 sequential values. These values were then organized in sets of 5 per row, with each following row offset by one value. This allowed for every combination of the sequence to be run through the network and made it possible for the network to be trained for all cases.

The training data was then normalized to be between 0 and 1. This was done by dividing each training set's values by 2^n, where n is the order of the primitive polynomial. The rows of data were then shuffled. 20% of the data was set aside for testing. 10% of the remaining data was then set aside for a validation data set. The training data that was

Table 3.2: Sample Training Data

Training Set	Sample t=-3	Sample t=-2	Sample t=-1	Current Sample	Next Value
1	236	146	223	147	83
2	146	223	147	83	48
3	223	147	83	48	24
4	147	83	48	24	202
5	83	48	24	202	52
6	48	24	202	52	191
7	24	202	52	191	162
8	202	52	191	162	199
9	52	191	162	199	89
10	191	162	199	89	103

used to train the final model consisted of a combination of all of the data used to train the individual models. This was also shuffled and broken up into training, test, and validation data in the same manner. The final neural network was trained over 1,500 epochs, batch size of 16 and a learning rate of 0.0001.

3.5 Simulation Model

The initial Matlab testing of this model consisted of exporting the model from Pytorch as an ONNX file, loading the ONNX file into Matlab, and then running the simulation. Within the simulation, it is assumed that this network has been notified that there is a frequency-hopping jammer present. Once the network is notified of this, transmission is paused for 4 jammer hops while the jammer frequencies are sampled and saved into memory. Once 4 hops are sampled, the simulation then feeds the sampled hops through the neural network, predicting where the 5th hop will be. After the fifth hop is predicted, the system then chooses a frequency to hop to that is 128 channels away from the predicted jamming frequency. Once 25 hops have passed, the network has an algorithm that is able to determine if the jammer is utilizing an m-sequence that it is trained for and exactly which primitive polynomial the jammer is utilizing. This information will be useful in the event that the jammer is using a sequence that the transmitting system is not prepared to defend against.

The simulation then compares the jamming frequency to the transmitting frequency at each time, and the probability of jamming P_{jam} was calculated as the ratio of the number of successful jamming instances Num_{Jams} to the overall number of hops $Total_{Hops}$ as illustrated in Eq. 3.1. To determine if the signal was successfully jammed, the output from the jammer and the transmitter m-sequence codes had to produce the same number on a given sequence call.

Simulations were also run where the jamming system has the prediction algorithm and the transmitter does not. Then, both the transmitter and jammer have it.

$$P_{jam} = \frac{Num_{Jams}}{Total_{Hops}} \tag{3.1}$$

The jammer was run using each of the different m-sequences that the model was trained for on each run. Then, a baseline transmitter was run using a different m-sequence than the jammer. This resulted in some of the transmitted signals being successfully jammed. The smart transmitter then waits for the first 4 jamming frequencies to be sampled and then starts to feed the previous 3 samples with the current through the neural network in order to predict what the next hop will be. Then, the selected frequency to hop to is offset from the predicted jamming frequency by half of the total number of channels, 128 in this case. If the selected channel is greater than 256, then the system wraps the chosen channel back starting at 1. So, if the predicted jamming channel was 210, then the offset

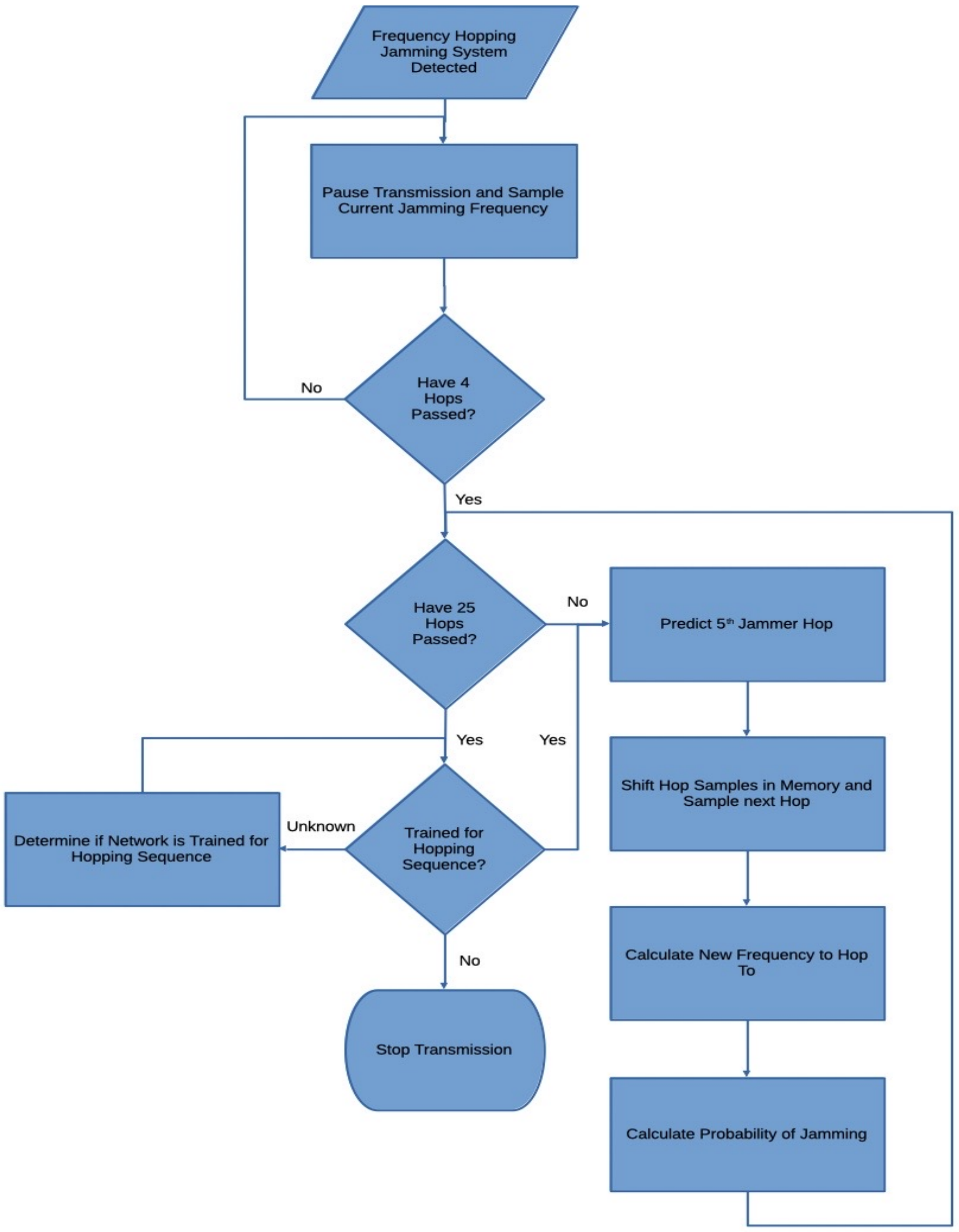

Frequency Hopping Jamming System Detected
Pause Transmission and Sample Current Jamming Frequency
Have 4 Hops Passed?
No
Yes
Have 25 Hops Passed?
No
Predict 5th Jammer Hop
Yes
Yes
Shift Hop Samples in Memory and Sample next Hop
Determine if Network is Trained for Hopping Sequence
Unknown
Trained for Hopping Sequence?
Calculate New Frequency to Hop To
No
Stop Transmission
Calculate Probability of Jamming

Figure 3.4: Simulation Flow Chart

channel would be 338, and the chosen channel would be 82 (or 338-256). This methodology proved to work well because the neural network was trained with an accuracy near 100%. This helped to reduce the jamming probability to 0 for all cases.

Once these cases were run, the jammer was then given the neural network so that it can predict where the transmitting system will hop to. For this case, the transmitting system uses the pre-trained m-sequences. Then, the jammer continues to use the neural network and the transmitter also uses the neural network to demonstrate how the system performs in the presence of a smart jamming system.

3.6 Check for Pre-Trained Polynomial

While the system is actively predicting and choosing a hopping sequence based on the predicted jamming frequency, the system is saving the first 25 hops into memory to run through a prediction algorithm that can accurately determine if the jamming hopping sequence is something that the network is already trained for. In the unlikely event that it is not, the transmitting will cease communication until the jamming signal has stopped transmitting before transmitting with the original hopping sequence defined by the system.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 NEURAL NETWORK OPTIMIZATION - 12TH ORDER POLYNOMIAL

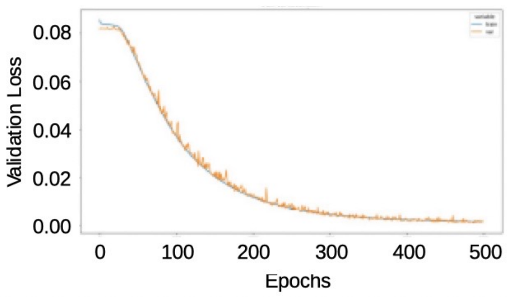
The neural network optimization task was focused on the 12th-order polynomial and started with 4 inputs and 1 output and then varied the number of hidden layers, the width of the hidden layers, different activation functions, and different numbers of training sets. As expected, it was found that the more data available to run through the model, the higher the accuracy. It needs at least one full sequence to run through the model as training data in order for the network to accurately learn the pattern of the sequence. For the activation functions, ReLU was the primary activation function used. Sigmoid was tried but did not provide nearly the same accuracy as the ReLU, so that was short-lived. Ultimately, the neural network that was chosen contained 5 hidden layers, each containing 256 neurons. This allowed for maximizing the accuracy of the system while minimizing the total computations required.

4.1.1 NUMBER OF HIDDEN LAYERS

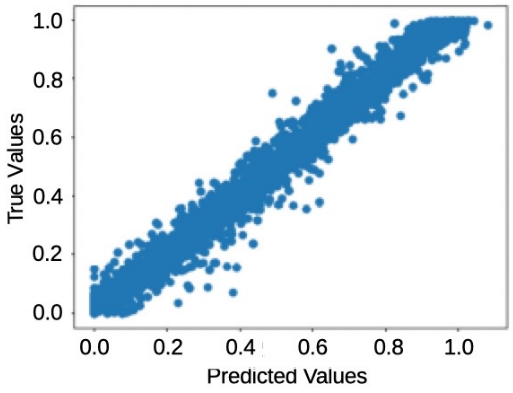
To test the effect of the number of hidden layers on the learning ability of the neural network for the 12th order polynomial, the number of layers was varied between 4, 5, and 8 hidden layers, shown in Figure 4.1. All of these have 256 wide layers, and the case with 5 layers had the best performance, as shown in Table 4.1.

4.1.2 VARYING THE WIDTH

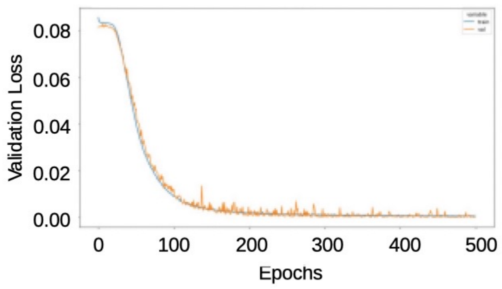
To test the effect of the number of neurons in each hidden layer on the learning ability of the neural network for the 12th-order polynomial, the number of neurons in each layer was varied between 64, 128, and 256 neurons, shown in Figure 4.2. All of these have 8



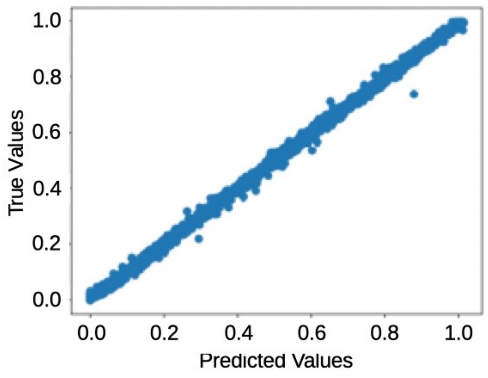
(a) 4 Layers - Training Results



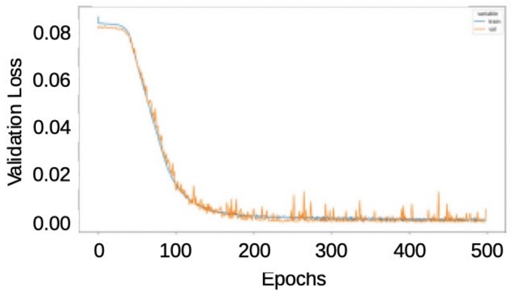
(b) 4 Layers - Test Results



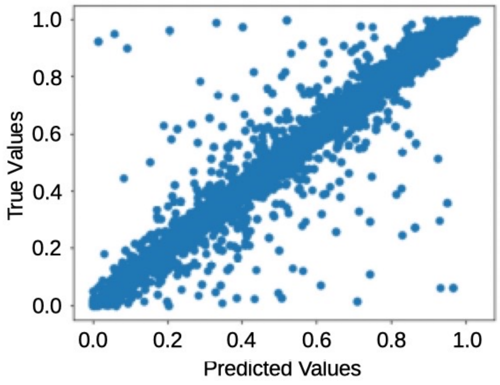
(c) 5 Layers - Training Results



(d) 5 Layers - Test Results



(e) 8 Layers - Training Results



(f) 8 Layers - Test Results

Figure 4.1: Comparison of Different Numbers of Layers

Table 4.1: Number of Layers Comparison

Number of Layers	R^2	MSE
4	0.9795	0.0017
5	0.9987	0.00011
8	0.9255	0.0062

hidden layers, and the case with 128 neurons had the best performance, as shown in Table 4.2.

Table 4.2: Number of Layers Comparison

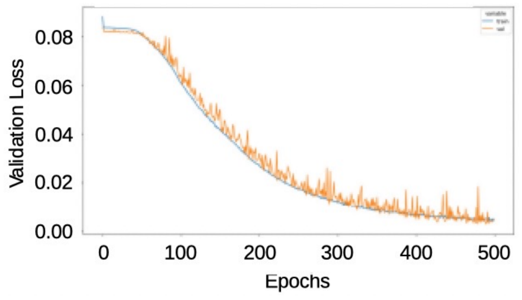
Width of Layers	R^2	MSE
64	0.9461	0.0045
128	0.9985	0.00013
256	0.9255	0.0062

4.2 Neural Network Training and Test Results - Individual Polynomials

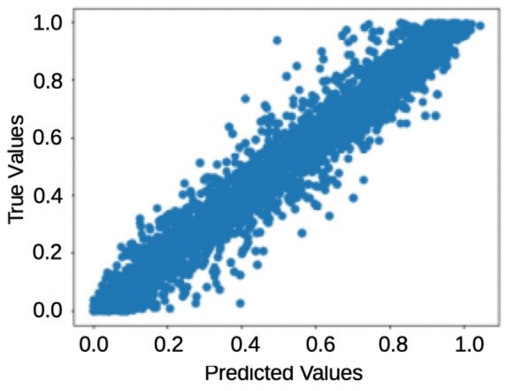
After the neural network optimization was completed, the chosen network was then trained individually for each of the chosen polynomials with a very high degree of accuracy. The results from this can seen in Figure 4.3, Figure 4.4, and Table 4.3.

4.3 Final Network Training Results

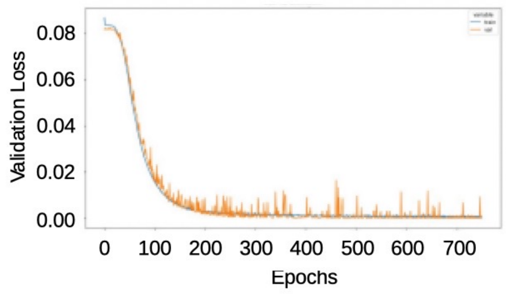
Figure 4.5 shows the training results for the final neural network that is able to predict any of the 7 m-sequences used. Very good accuracy is also shown here, with a total mean squared error (MSE) training loss of 0.00027 and an MSE validation loss of 0.00015.



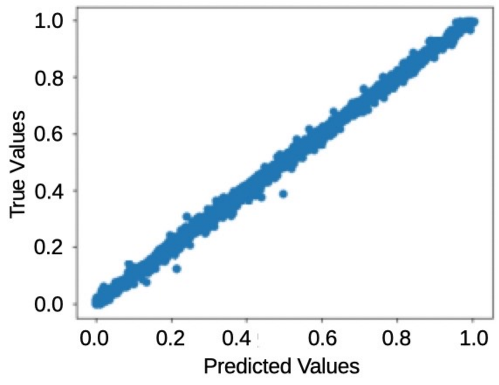
(a) 64 Width - Training Results



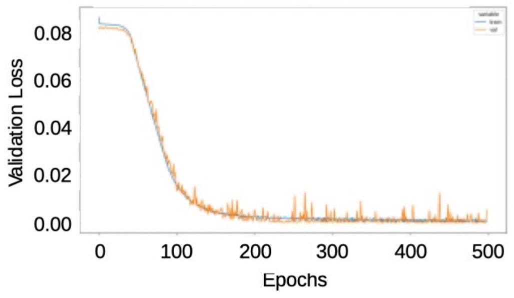
(b) 64 Width - Test Results



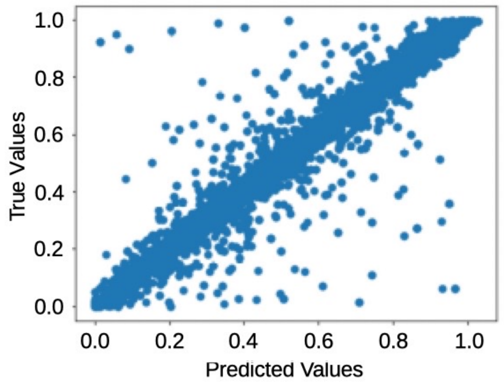
(c) 128 Width - Training Results



(d) 128 Width - Test Results



(e) 256 Width - Training Results



(f) 256 Width - Test Results

Figure 4.2: Comparison of Different Widths

Table 4.3: Neural Network Training Results

Order	R^2	MSE
6	0.9999	$3.034 * 10^{-6}$
7	0.9996	$3.096 * 10^{-5}$
8	0.9999	$1.211 * 10^{-5}$
9	0.9999	$2.538 * 10^{-6}$
10	0.9996	$3.232 * 10^{-5}$
11	0.9997	$2.151 * 10^{-5}$
12	0.9987	$1.118 * 10^{-4}$

Figure 4.6 shows the training and test results for the final neural network that is able to predict any of the 7 m-sequences used. Very good accuracy is also shown here. When tested, the neural network had an R^2 value of 0.99817.

4.4 Simulation Results

Table 4.5 shows the overall results for the network. The jamming order is the order of the primitive polynomial that was chosen from Table 3.2 for the jamming m-sequence. The transmitting order is the order of the primitive polynomial chosen from Table 3.2. P_{jam} is the calculated probability of jamming without the neural network predicting the jammer hopping sequence and telling the transmitting system where to hop. $P_{jamSmart}$ is the calculated probability of jamming with the neural network predicting the jamming hopping sequence and telling the transmitting system where to hop. It is shown that for all tested cases, the probability of jamming was reduced to 0 when the neural network is in the loop predicting the next hop for the jamming system.

Figure 4.7 shows the accuracy of the final neural network's accuracy when imported and implemented into the Matlab system model. Near-perfect accuracy is demonstrated here for each case. It is shown that similar results are achieved to the initial training and testing conducted in Pytorch.

The results here show that the system is highly accurate in predicting a jamming signal that uses a hopping sequence that the neural network has been trained for. Because of this, this system can choose hopping sequences that completely eliminate the interference caused by a frequency hopping jammer. Once 25 hops have passed, the network can also determine with a 96.83% degree of accuracy if the jammer is using an m-sequence the network is trained for and which one.

Table 4.4: Experimental Results for Probability of Jamming with/without the Smart Hopping Sequence

Case No.	Jammer Order	Transmitting Order	P_{jam}	$P_{jamSmart}$
1	6	9	0.003265	0
2	7	9	0.003912	0
3	8	9	0.003879	0
4	9	9	0.001957	0
5	10	9	0.003906	0
6	11	9	0.003899	0
7	12	9	0.003956	0

4.4.1 Smart Jamming System

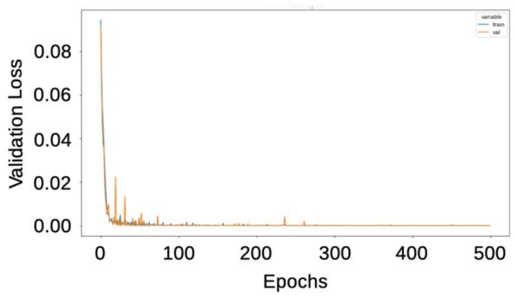
The results here show that when the jamming system utilizes the trained neural network for the transmitting hopping sequence, the transmitting system can be jammed between 11% and 12% of the time. However, when the neural network is also added to the transmitting system, the probability of jamming is greatly reduced to 0.002441, or about 0.24%.

Table 4.5: Experimental Results for Probability of Jamming with a Smart Hopping Jamming System

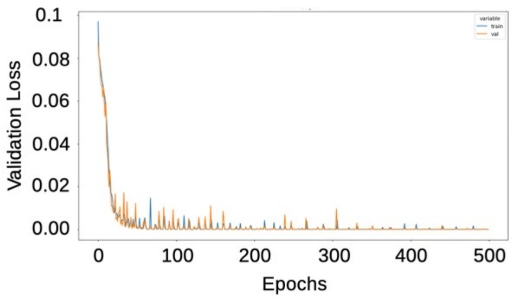
Case No.	Transmitting Order	P_{jam}
1	6	0.2063
2	7	0.1181
3	8	0.1844
4	9	0.2035
5	10	0.1769
6	11	0.1856
7	12	0.1731

4.4.2 Changing the Seed

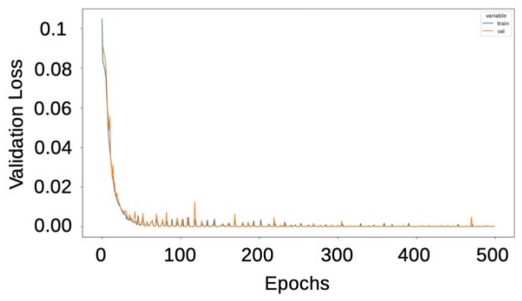
For all cases when a different seed is used for the starting point of the m-sequence, there was no change in the simulation results because the network was trained to predict the sequence regardless of the starting point as long as the sample used for prediction consists of a sequence of 4 consecutive hops with the same seed.



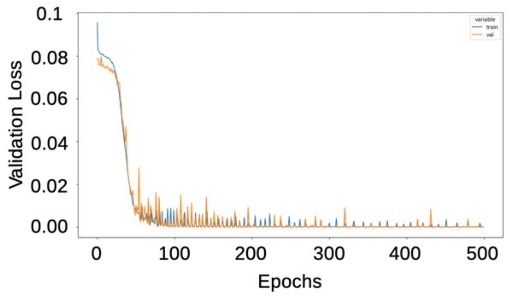
(a) 6th-Order



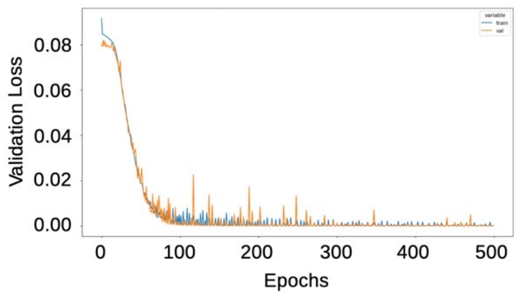
(b) 7th-Order



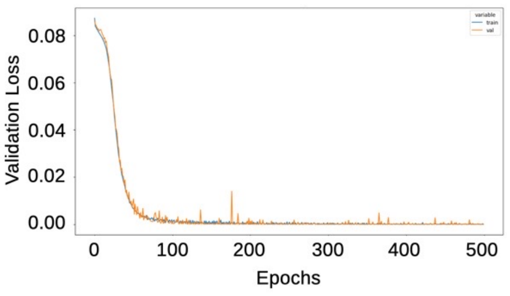
(c) 8th-Order



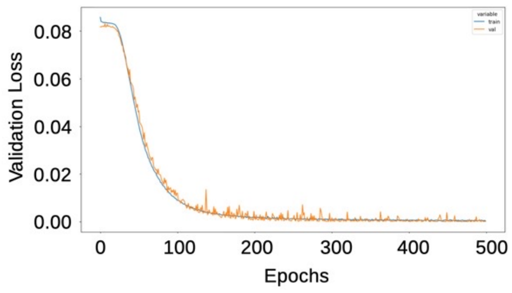
(d) 9th-Order



(e) 10th-Order

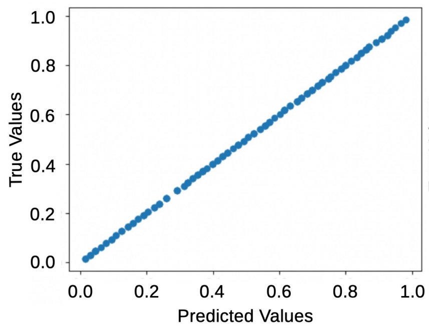


(f) 11th-Order

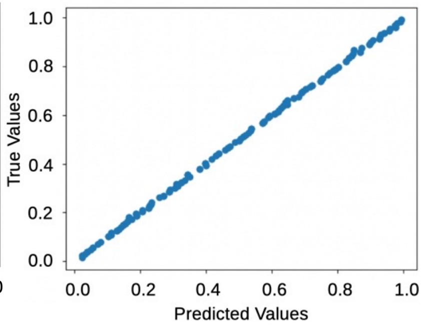


(g) 12th-Order

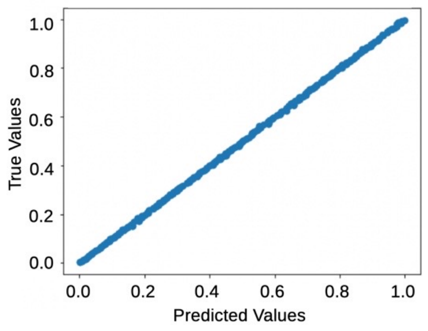
Figure 4.3: Training Test Results for the various order polynomials



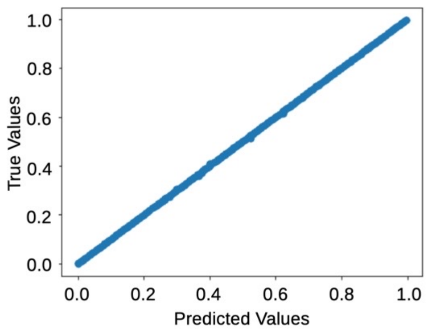
(a) 6th-Order



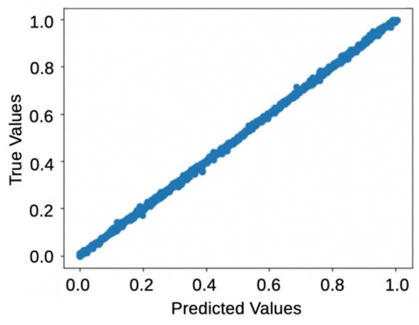
(b) 7th-Order



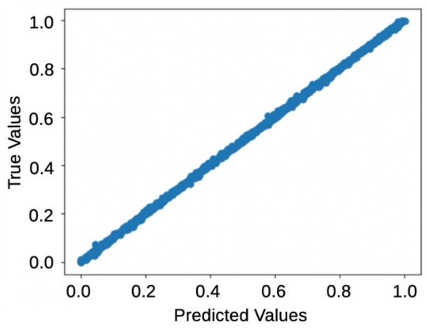
(c) 8th-Order



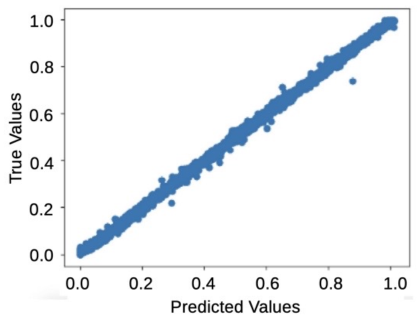
(d) 9th-Order



(e) 10th-Order



(f) 11th-Order



(g) 12th-Order

Figure 4.4: Training Test Results for the various order polynomials

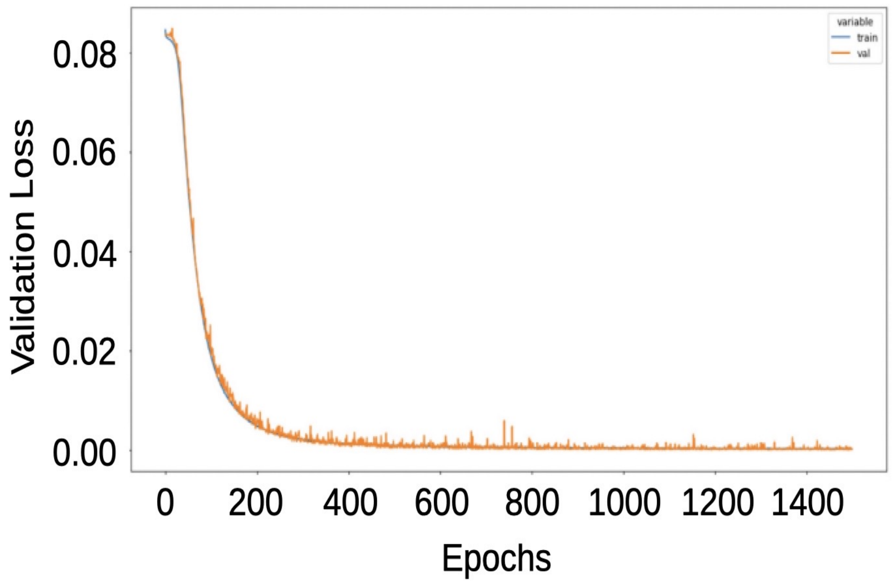


Figure 4.5: Neural Network Training Results

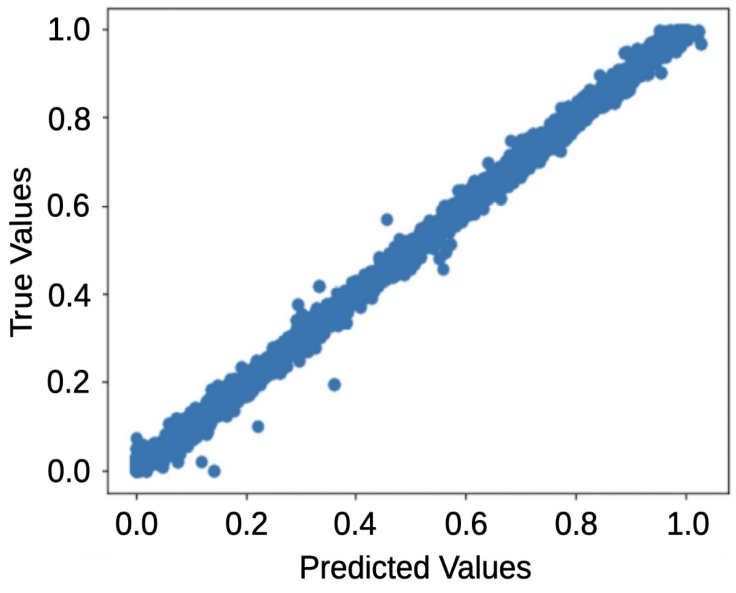
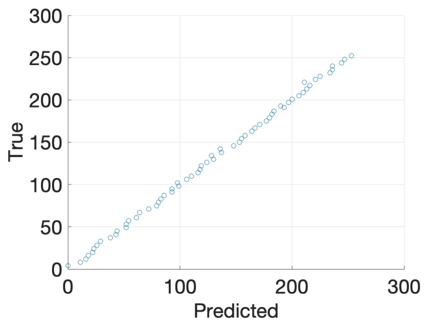
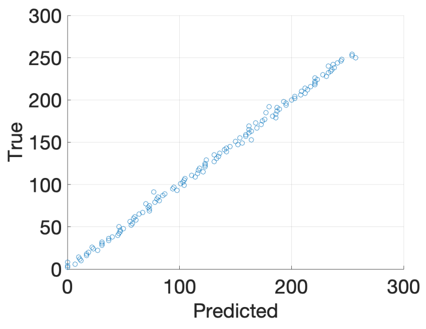


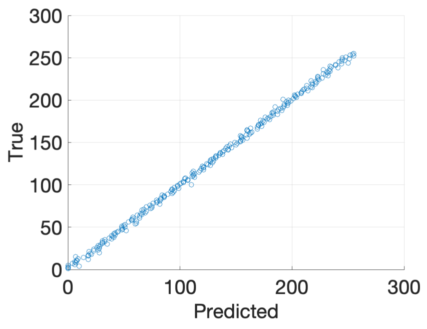
Figure 4.6: Neural Network Test Results



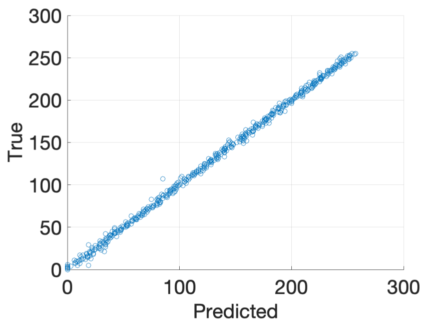
(a) 6th-Order



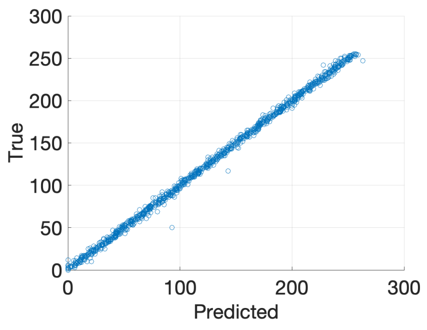
(b) 7th-Order



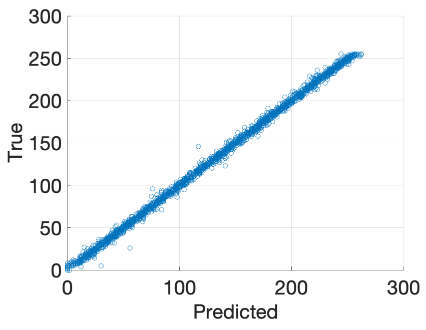
(c) 8th-Order



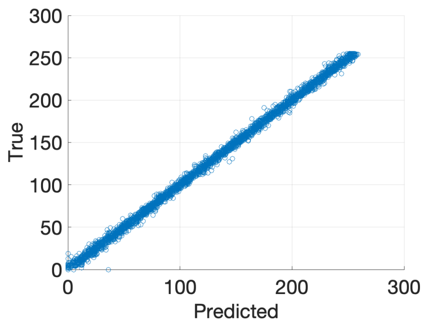
(d) 9th-Order



(e) 10th-Order



(f) 11th-Order



(g) 12th-Order

Figure 4.7: Predicted vs. True plots for the various order polynomials

CHAPTER 5
CONCLUSION

Jamming systems can be detrimental to the signal integrity and transmission security of radio frequency systems. While certain methods exist in order to combat it, like frequency hopping, it is very difficult to develop a system that is immune to these jamming attacks. One type of jamming system that mimics noise is a noise jammer that hops over various carrier frequencies.

In this work, a novel approach was proposed to use a neural network to predict where a frequency-hopping jammer will hop. Once the jamming signal's presence is known and being tracked, the RF system can then use the network presented here to choose a hopping sequence that completely eliminates the interference caused by the jamming signal.

The artificial neural network presented in this work has the capacity to learn multiple PN sequences and can predict the next value in the sequence with a high degree of accuracy. This network consists of an input layer with 4 neurons, 4 hidden layers, each containing 1024 neurons, and an output layer with a single output neuron. This network was trained for 7 different m-sequences, whose primitive polynomials were orders 6 through 12. The primitive polynomial for each sequence was chosen as a minimal polynomial. An algorithm was also developed that allows for the system to determine if the jamming system is using a primitive polynomial that is known by this system.

By completely eliminating the jamming interference of a frequency-hopping jammer, this network architecture allows for increased transmission security in frequency-hopping systems. Similar work can also be done to predict any PRNG with a sufficient amount of training data. This has the potential to predict more complex frequency hopping schemes and even has applications outside of these sequences, like data encryption.

This work can be expanded in the future to create networks that are trained for more primitive polynomials of higher order, different PRNGs, and even networks that can calculate the polynomial used in the sequence generation when the network has not been trained for a specific polynomial. Given that this network has to have an entire sequence run through it in order to recognize the pattern, future work could also be conducted here to figure out how to develop a neural network that is able to characterize the pattern of more complex PRNGs without the need to run an entire sequence through the network. This would open up the possibilities for what they are capable of learning. The neural network presented here could be scaled to learn any PRNG sequence; the user would only be limited by the time and hardware available to them.

REFERENCES

[1] M. A. Soliman, K. H. Moussa, W. M. Saad, S. Shaaban, and M. R. M. Rizk, “Analysis of jamming attacks on a hopped ofdm communication system,” in *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, 2018, pp. 407–411.

[2] K. Grover, A. Lim, and Q. Yang, “Jamming and anti-jamming techniques in wireless networks: A survey,” *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 17, p. 197, 01 2014.

[3] Y. Arjoune, F. Salahdine, M. S. Islam, E. Ghribi, and N. Kaabouch, “A novel jamming attacks detection approach based on machine learning for wireless communication,” in *2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 459–464.

[4] J. Price, Y. Li, K. A. Shamaileh, Q. Niyaz, N. Kaabouch, and V. Devabhaktuni, “Real-time classification of jamming attacks against uavs via on-board software-defined radio and machine learning-based receiver module,” in *2022 IEEE International Conference on Electro Information Technology (eIT)*, 2022, pp. 1–5.

[5] H. Pirayesh and H. Zeng, “Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey,” *IEEE Communications Surveys Tutorials*, vol. 24, no. 2, pp. 767–809, 2022.

[6] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, “A survey on jamming attacks and countermeasures in wsns,” *IEEE Communications Surveys Tutorials*, vol. 11, no. 4, pp. 42–56, 2009.

[7] R. D. Pietro and G. Oligeri, “Jamming mitigation in cognitive radio networks,” *IEEE Network*, vol. 27, no. 3, pp. 10–15, 2013.

[8] M. Annulli. Noise jamming (radar). [Online]. Available: https://www.emsopedia.org/entries/noise-jamming-radar/

[9] K. N. Vaishnavi, S. D. Khorvi, R. Kishore, and S. Gurugopinath, “A survey on jamming techniques in physical layer security and anti-jamming strategies for 6g,” in *2021 28th International Conference on Telecommunications (ICT)*, 2021, pp. 174–179.

[10] Y.-B. Luo, B.-S. Wang, and G.-L. Cai, "Effectiveness of port hopping as a moving target defense," in *2014 7th International Conference on Security Technology*, 2014, pp. 7–10.

[11] Y. Wang, Y. Niu, J. Chen, F. Fang, and C. Han, "Q-learning based adaptive frequency hopping strategy under probabilistic jamming," in *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, 2019, pp. 1–7.

[12] M. O. Mughal and S. Kim, "Signal classification and jamming detection in wide-band radios using naïve bayes classifier," *IEEE Communications Letters*, vol. 22, no. 7, pp. 1398–1401, 2018.

[13] M. Ren, B. Cheng, and P. Gao, "Active jamming signal recognition based on residual neural network," in *2022 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, 2022, pp. 1–4.

[14] R. Morales Ferre, P. Richter, A. De La Fuente, and E. Simona Lohan, "In-lab validation of jammer detection and direction finding algorithms for gnss," in *2019 International Conference on Localization and GNSS (ICL-GNSS)*, 2019, pp. 1–6.

[15] Q. Wang, "Optimal sets of frequency hopping sequences with large linear spans," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1729–1736, 2010.

[16] H. Lv, J.-C. Fang, J.-X. Xie, and P. Qi, "Generating of a nonlinear pseudorandom sequence using linear feedback shift register," in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 432–435.

[17] S. S. Mansouri and E. Dubrova, "An improved hardware implementation of the grain stream cipher," in *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, 2010, pp. 433–440.

[18] J. Lan, W. L. Goh, Z. H. Kong, and K. S. Yeo, "A random number generator for low power cryptographic application," in *2010 International SoC Design Conference*, 2010, pp. 328–331.

[19] Y. Wang, Y. Hao, H. Han, J. Lu, and H. Li, "Frequency hopping pattern and synchronization based on hopping spread spectrum communication," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, 2021, pp. 894–898.

[20] J. Komo and S.-C. Liu, "Frequency hopping m-sequences," in *Proceedings. IEEE Energy and Information Technologies in the Southeast'*, 1989, pp. 855–859 vol.2.

[21] Z. Syroka, T. Zajac, and P. Dubiłowicz, "Generation of linear maximum length sequences," in *2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, 2010, pp. 309–313.

[22] Chin and McCluskey, "Test length for pseudorandom testing," *IEEE Transactions on Computers*, vol. C-36, no. 2, pp. 252–256, 1987.

[23] E. Eröz, E. Tanyıldızı, and F. Özkaynak, "Determination of suitable configuration parameters for linear feedback shift register using binary bat optimization algorithm," in *IEEE EUROCON 2021 - 19th International Conference on Smart Technologies*, 2021, pp. 348–351.

[24] L.-T. Wang and E. McCluskey, "Hybrid designs generating maximum-length sequences," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 91–99, 1988.

[25] W. Stahnke, "Primitive binary polynomials," *Mathematics of Computation*, vol. 27, no. 124, pp. 977–980, 1973.

[26] mathworks.com. Pn sequence generator. [Online]. Available: https://www.mathworks.com/help/comm/ref/pnsequencegenerator.html

[27] liquidsdr.org. Maximal-length sequence generator (msequence). [Online]. Available: https://liquidsdr.org/doc/msequence/

[28] W. Jin, J. Wang, H. Liao, and L. Gan, "Estimation of primitive polynomial for m-sequence with finite sequence length," in *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, 2018, pp. 1–4.