2022-09

MAXIMAL-LENGTH SEQUENCE CODE CLASSIFICATION OPTIMIZATION PROCEDURE UTILIZING DEEP LEARNING NEURAL NETWORKS

Reeder, Christina N.

Monterey, CA; Naval Postgraduate School

https://hdl.handle.net/10945/73049

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MAXIMAL-LENGTH SEQUENCE CODE CLASSIFICATION OPTIMIZATION PROCEDURE UTILIZING DEEP LEARNING NEURAL NETWORKS

by

Christina N. Reeder

September 2022

Thesis Advisor:	Ric Romero
Second Reader:	Roberto Cristi

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE MAXIMAL-LENGTH SEQUENCE CODE CLASSIFICATION OPTIMIZATION PROCEDURE UTILIZING DEEP LEARNING NEURAL NETWORKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Christina N. Reeder	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) NAVAIR, Patuxent River, MD 20670	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Direct sequence spread spectrum techniques are often utilized in encoding communications signals because they can decrease signal spectrum lower than the thermal noise floor of a receiver, making them harder to detect. Accurate and timely classification of spreading codes for message decoding has become an area of interest. In this work, we evaluate the difference in classification performance between a traditional matched filter bank method and trained neural networks. We demonstrate that trained neural networks may outperform matched filters specifically in the medium SNR range. Additionally, we explore performance of a neural network trained to detect and classify direct sequence coded signals along with a null alternative by adding a "noise only" signal classification option. We find that there is a probability of false alarm (P_{fa}) associated with a neural network trained to detect signals with a "noise only" classification option. We conclude that trained neural networks offer an increase in both percentage of classification (P_c) and time-to-classify performance. However, we also conclude that more work is required to optimize the neural network for the decoding of preamble codes of different lengths and types. This work demonstrates the feasibility of using trained neural networks for use in decoding direct sequence coded signals.

14. SUBJECT TERMS machine learning, signal detection, signal classification, maximal-length sequence, spread spectrum, deep learning, neural network	**15. NUMBER OF PAGES** 63
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MAXIMAL-LENGTH SEQUENCE CODE CLASSIFICATION OPTIMIZATION PROCEDURE UTILIZING DEEP LEARNING NEURAL NETWORKS

Christina N. Reeder
Lieutenant, United States Navy
BS, Colorado School of Mines, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2022

Approved by: Ric Romero
Advisor

Roberto Cristi
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Direct sequence spread spectrum techniques are often utilized in encoding communications signals because they can decrease signal spectrum lower than the thermal noise floor of a receiver, making them harder to detect. Accurate and timely classification of spreading codes for message decoding has become an area of interest. In this work, we evaluate the difference in classification performance between a traditional matched filter bank method and trained neural networks. We demonstrate that trained neural networks may outperform matched filters specifically in the medium SNR range. Additionally, we explore performance of a neural network trained to detect and classify direct sequence coded signals along with a null alternative by adding a "noise only" signal classification option. We find that there is a probability of false alarm (P_{fa}) associated with a neural network trained to detect signals with a "noise only" classification option. We conclude that trained neural networks offer an increase in both percentage of classification (P_c) and time-to-classify performance. However, we also conclude that more work is required to optimize the neural network for the decoding of preamble codes of different lengths and types. This work demonstrates the feasibility of using trained neural networks for use in decoding direct sequence coded signals.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AF	activation function
DOD	Department of Defense
HL	hidden layer
m-sequence	maximal-length sequence
NaN	not-a-number
NPS	Naval Postgraduate School
P_c	percentage of classification
P_{fa}	probability of false alarm
ReLU	*rectified linear unit*
SNR	signal-to-noise ratio
USN	U.S. Navy

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank my thesis advisor, Professor Romero, for taking me on as one of his thesis students. His guidance and expertise allowed me to accomplish a scope of research I am very proud to put my name to.

I would also like to thank my parents Steve and Kelly. From the time I said I was joining the Navy 12 years ago you've been my biggest cheerleaders. From the first day of flight training to now, graduating with my master's, you've always been there for me, and I'm so grateful to call you my parents.

I would also like to acknowledge my daughter, Natalija. Your presence in my life made me strive to accomplish something that will make you proud to have me as your mom. And to my husband Jim, whose constant support helped me strive for success throughout my time here at NPS. I wouldn't want anyone else as my partner on this crazy journey. I love you both very much.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

Encoding of communications signals is, tactically, an important aspect of military operations. Sending encoded signals that either the enemy cannot find amidst background noise or that are difficult to decode is the key to secure communications between assets. However, conversely, our capacity to find enemy signals and decode them efficiently can lead to intelligence gathering that allows for effective and timely decision making.

The automation and optimization of the detection and decoding of communications transmissions via deep learning can have a profound impact on our war-fighting ability. This paper proposes that a neural network trained in the classification of communications spreading codes or preambles may, in some cases, outperform the traditional methods used in practice. Neural networks, if trained properly, can classify signals more quickly and accurately, allowing for more expedient intelligence gathering capabilities.

1.1 Thesis organization

This thesis is organized in a manner to provide background material in Chapters 2 and 4, while our methods and findings are presented in Chapters 3, 5, and 6. This thesis begins with traditional methods for communications signals classification and then discusses the use of trained neural networks. For each classification method investigated, the assumptions are stated clearly to set up each training and testing experiment.

We begin by reviewing the foundation of spread spectrum techniques in Chapter 2. This chapter specifically dives into maximal-length sequence (m-sequence) signals, as these have unique properties as opposed to other code families. We then look into the traditional method of classifying m-sequences through the use of matched filters in Chapter 3.

Chapter 4 inspects the elements of deep learning via a neural network. This thesis focuses on three variations of neural networks and the differences between them. Chapter 5 is a deep investigation into the parameterization of different variables for the neural networks. We begin by: a) comparing different activation function (AF) choices; b) deciding on the number of nodes in a hidden layer (HL); and c) varying the neural network training rate. We conclude the chapter with a performance comparison between the traditional matched filter bank method discussed in Chapter 3 and the neural networks discussed in Chapter 5.

The final chapter is an investigation into the feasibility of using neural networks to determine if a m-sequence coded signal is present in noise along with a "noise-only" alternative, as opposed to assuming the signal's presence in the received measurement. Chapter 6 looks at the differences in percentage of classification (P_c) between the neural networks trained to find the m-sequence coded signal in noise and the neural networks that assumed the coded signal is already present. We then conclude by investigating the probability of false alarm (P_{fa}) for neural networks trained to distinguish between noise and the presence of a m-sequence coded signal.

CHAPTER 2:
Spread Spectrum Modulation

Spread spectrum utilizes a digital modulation scheme in which a signal with a small bandwidth is converted to one with a wide bandwidth [1]. In military applications, defense against jamming and message interception are high priorities when sending and receiving communications signals. Spread spectrum signals are a very common form of bandwidth expansion, which forces potential jammers to cover wider frequency spectra, thereby diluting their effectiveness [1]. In the realm of covert communications, the purpose of spread spectrum modulation is to provide the signal with a low spectral level, so that it is masked by background noise, thus making it harder to intercept [2].

2.1 Maximal-length sequences

The key to receiving or intercepting spread spectrum communications is knowledge of the code used to encode the signal. M-sequences are a commonly utilized direct sequence code, with very specific properties that are often created via linear feedback shift-register method [2]. The creation of new and longer m-sequence codes is the topic of many research papers, good examples of which are found in these reference papers [3]–[7]. Codes that are longer in length allow for more possible codes, while still following the properties of m-sequences as stated in Subsection 2.1.1.

2.1.1 Properties of m-sequences

The specific properties given in [8] and [2] are what define m-sequences. The autocorrelation property of m-sequences, which is presented in [8], is what makes this modulation scheme particularly useful for communication signals. The periodic autocorrelation function, $\theta_b(k)$ is seen in Equation 2.1 [8]. In this equation N is the number of chips in the m-sequence, k corresponds to the k^{th} cyclic shift of the sequence, and l is any integer.

$$\theta_b(k) = \begin{cases} 1.0 & k = lN \\ -\frac{1}{N} & k \neq lN \end{cases} \tag{2.1}$$

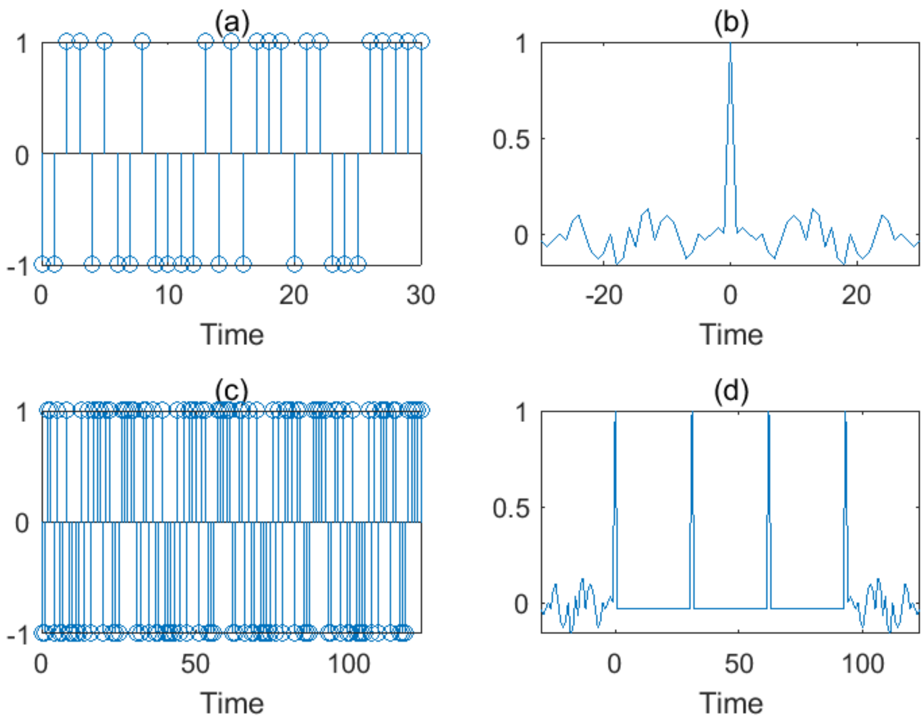

Figure 2.1. M-sequence autocorrelation property: (a) single 31-bit m-sequence, (b) single m-sequence autocorrelation, (c) 4 repeated m-sequences, and (d) repeated m-sequences' autocorrelation.

The periodic autocorrelation property of m-sequences is illustrated in Figure 2.1(d) using four repeated sequences. Figure 2.1(a) and (b) show a single 31-bit m-sequence and its autocorrelation function. The autocorrelation function allows a visual representation of how closely a signal matches a copy of itself shifted in time [9]. As seen in Figure 2.1(b), the autocorrelation function of a single m-sequence is symmetrical in time about zero delay, and the maximum value occurs at time zero delay [9]. Figure 2.1(c) and (d) show the same 31-bit m-sequence repeated four times, and the autocorrelation function for the repetitive signal. This practical autocorrelation plot matches Equation 2.1 for $\theta_b(k)$. This autocorrelation function has peaks at time 0, 31, 62, and 93, all of which are the locations at which the repetitive m-sequences begin.

2.2 Detection of spread spectrum signals

In practice, communication signals are passed through a multi-layer demodulation system. These demodulators, generally, have some form of energy detector as the first layer that locates a signal in the noise. After an m-sequence coded signal is located (in time), then comes the task of actually deciphering the spreading code used such that the bits or symbols in a message can be decoded. When a received signal is expected and the code is known, this is a simple process. However, when performing covert operations in which the signal is intercepted and the code is not known, determining which preamble or spreading code is utilized to be able to detect/locate or decipher the signal is a more laborious task. There are many research papers that attempt to optimize the detection and demodulation of spread spectrum signals. Good examples of which are found in these reference papers [10]–[14].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Matched Filter Classification of M-Sequences

Once a coded signal is located amidst the noise, it must then be decoded so the sent message can be read. To decode the signal, the preamble, or code used to encode the message must be known so the proper algorithm can be utilized. One method of decoding is through the use of a bank of matched filters corresponding to the many possible signals, such as the six m-sequences applied in this thesis.

3.1 The matched filter

A filter specifically designed to correlate exactly to a transmitted signal is known as a matched filter. Reference [15] goes through the theory behind the development of matched filter, ultimately leading to Equations 3.1 and 3.2. These equations give the discrete-time equivalent to the analog matched filter and its discrete correlation output equivalent.

$$h[n] = ks^*[N_0 - n] \tag{3.1}$$

$$\begin{aligned} y[n] &= \sum_{m=-\infty}^{\infty} h[m]s[n-m] \\ &= \sum_{m=-\infty}^{\infty} s[m]s^*[m + N_0 - n] \end{aligned} \tag{3.2}$$

In Equation 3.1, $h[n]$ is the discrete time matched filter associated with the signal $s[n]$, where $n = 0, 1, 2, ...$ is the discrete-time index. The variable N_0 is the discrete-time length of the signal, and k is a constant [15]. As Equation 3.1 illustrates, the matched filter $h[n]$ is created via the complex conjugate of the time-flipped version of the signal expected to be received.

In Equation 3.2, the variable $y[n]$ is the discrete time output of the convolution between the matched filter and the input signal (i.e., the output signal when the input signal is filtered through the matched filter) where m is the temporary time index.

Note that the matched filter is only designed for one expected received signal. In the tactical realm of communications, each friendly force can be utilizing a different spreading code for its message transmission. Therefore, our receiver and its decoding logic must account for all possible codes to identify the preamble sequence being utilized.

In addition, in an interception operation, we are not the intended receiver of the signals being decoded. Therefore, our receiver has no prior knowledge of the code being used by the adversary transmitter. Just like in the case of friendly transmissions, a signal processing technique to classify which code is used is also needed.

3.2 Using matched filters bank to find m-sequence codes

Prior knowledge of all possible preamble codes being utilized is required when applying matched filters for decoding. Because of the specific properties of m-sequences, discussed in Section 2.1.1, possible sequences of a given length can be generated. We can therefore create a matched filter corresponding to each possible m-sequence of our chosen length. Then the received signal is convolved with the bank of matched filters corresponding to all the known codes being used. The matched filter with the output containing the largest peak value must correspond to one of the m-sequences being transmitted, which is then deemed to be the detected signal.

Table 3.1. M-sequences utilized in experiments

Code Number	M-sequence
1	0011010010000101011101100011111
2	0001101110101000010010110011111
3	0111000101011010000110010011111
4	0010011000010110101000111011111
5	0110011100001101010010001011111
6	0100010010101100001110011011111

Table 3.1 shows the six possible fifth degree m-sequence codes, that are each 31-bits in length. These codes are utilized in both the matched filter bank and neural network classification of input signals in this thesis.

3.2.1 Monte Carlo simulation

Our first experiment utilizes the matched filter bank approach to determine the m-sequence code present in the received signal. For this experiment, we assume the received m-sequence is one of the six codes in Table 3.1. Reference [16] describes the standard method utilized in a Monte Carlo simulation. Our simulation has 10,000 trials and a signal-to-noise ratio (SNR) ranging from -20 dB to 20 dB. For each trial, the received signal passes through each matched filter in the bank for each SNR. The classification for each trial is then decided by the matched filter output with the highest peak.

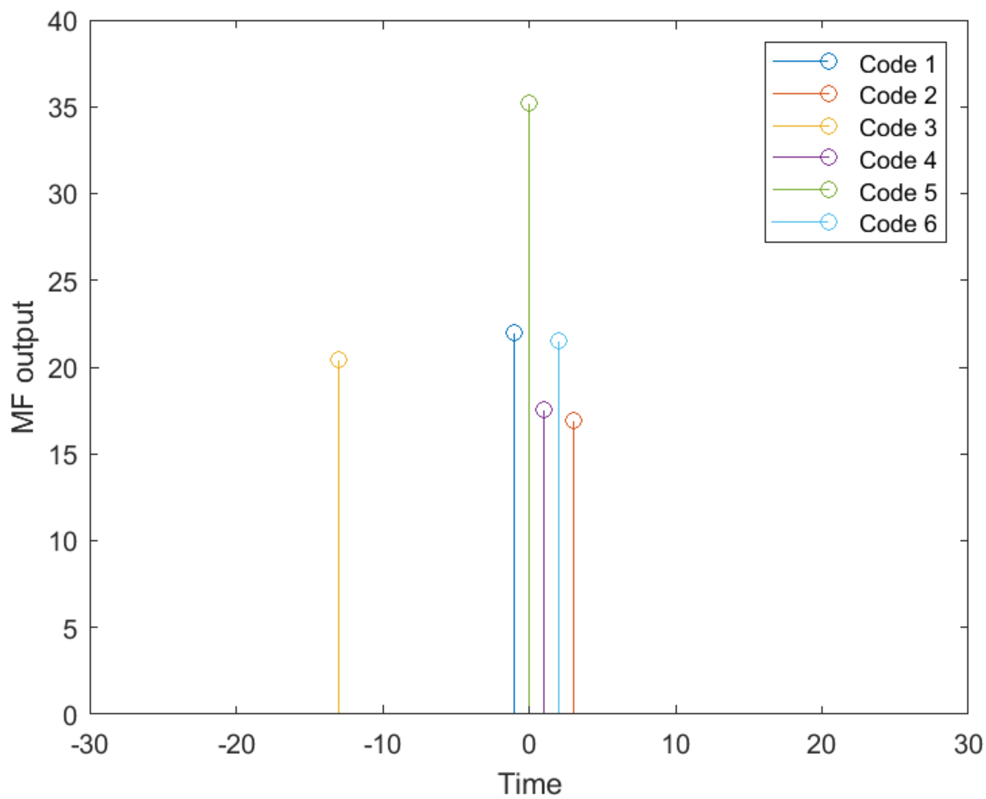

Figure 3.1. Maximum matched filter outputs from single trial

Figure 3.1 shows the maximum value for the six matched filter outputs of a single trial at a SNR of 10 dB. In this trial, the actual signal code is number 5, which also corresponds to the matched filter containing the largest peak. Therefore, this trial is noted to have made a correct spreading code classification in the Monte Carlo simulation. The ratio between the number of times in which the classifier made the correct decision to the total number of trials for each SNR in the simulation is deemed to be the percentage of classification performance for the matched filter bank method.

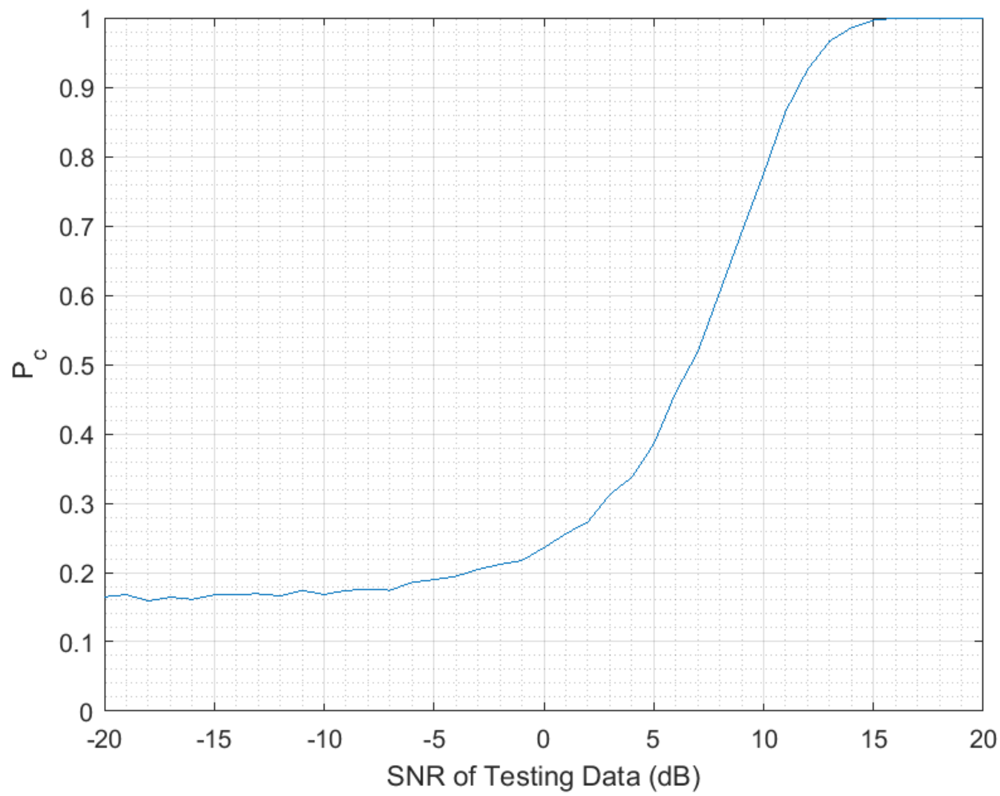

Figure 3.2. Monte Carlo results for classification via matched filter for randomly selected code

Figure 3.2 shows the percentage of classification (P_c) curve for our simulation. For these simulations, we assume that the m-sequence code signal is already present and that signal matches one of our six m-sequences. The m-sequence corresponding to the code is randomly selected from one of the six possible codes at the start of the simulation.

Figure 3.2 shows that at SNR values from -20 dB to -7 dB there is approximately a 17%

chance that the matched filter bank method is able to classify the m-sequence utilized to encode the signal. It is not until an SNR of approximately 7dB that the matched filter bank method has above a 51.79% P_c.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Machine Learning with Neural Networks

Machine learning utilizes neural networks to classify outcomes based on training received. Machine learning and neural networks are fundamental building blocks of artificial intelligence, i.e., our ability to train computers to make decisions without direct human input post training. The possibilities are endless when it comes to training machines to perform calculations and make decisions autonomously. Trained neural networks are able to make decisions based on calculations that are completed much faster than humans are able to complete the same type of calculations.

Machine learning optimization and automation for various tasks are the topics of many research papers. The evaluation of various techniques for more robust classification algorithms will lead to optimization breakthroughs in the future [17]–[22].

4.1 Neural networks

Machine learning is achieved via neural networks. Neural networks are comprised of hidden layers (HL) that each have a set number of nodes. Each node receives inputs from the layer before, executes a calculation based on its training, and provides an output to all the nodes of the layer that comes after.

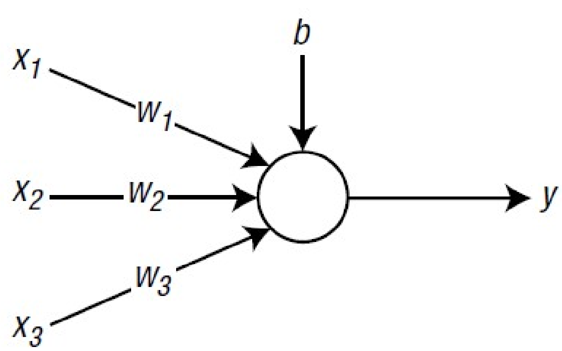

Figure 4.1. Visualization of neural network nodes. Source: [23]

Figure 4.1, from Reference [23], shows a visualization of how information flows into and out of these nodes. Each node calculates a weighted sum of its inputs and their respective weights given in vector form in Equation 4.1 [23], in which w is the vector of weights from the inputs, x is the values of each input, and b is the bias of that node.

$$v = wx + b \tag{4.1}$$

The output, v, of Equation 4.1 is then put through an activation function (AF) and used as the input to the nodes of the next layer in the neural network. The final output layer then provides the user with data based on the type of learning and application the neural network is being used for.

4.2 Supervised learning of the neural network

There are three types of machine learning: supervised, unsupervised, and reinforcement learning [23]. This paper utilizes supervised learning, in which labeled data is used to train the neural networks for signal classification. The other applications of supervised machine learning are regression and ranking. In regression, the output is a number within a set range, and in ranking, the output is a similarity value for verification in the regression [24].

Supervised learning with a classification application is used in this paper. With this type of learning, the network is trained via a set of inputs with a known output classification. Then when the trained network is presented with an input it has never seen before, the input is processed through the neural network and a classification decision is made. Given a test set with a significant number of inputs to the neural network, P_c can then be calculated for the classification accuracy of the network.

4.3 Hidden layers

Hidden layers are layers in the neural network not accessible from outside, i.e., their inputs and outputs are *hidden* from the user [23]. The number of hidden layers in a network and number of nodes in any given hidden layer is left to the discretion of the network designer.

There can be instances in which more layers or more nodes may be beneficial to help in the classification of complex inputs. However, the trade-off is that with each extra hidden layer and more nodes come a longer time to train the network due to the physical limits of the computer's processing power. This thesis research looks at the differences in classification performance of networks to classify signals with changes in the number of hidden layers, number of nodes, and different activation functions in the hidden layers.

This paper utilizes three different activation functions in the training and testing of neural networks for signal classification. The *sigmoid* and *rectified linear unit (ReLU)* functions are utilized in HL calculations, while the *softmax* AF is employed for the final P_c output.

4.3.1 *Sigmoid* AF

The first AF we investigate is the *sigmoid* function, given in Equation 4.2 [23], and shown in Figure 4.2.

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

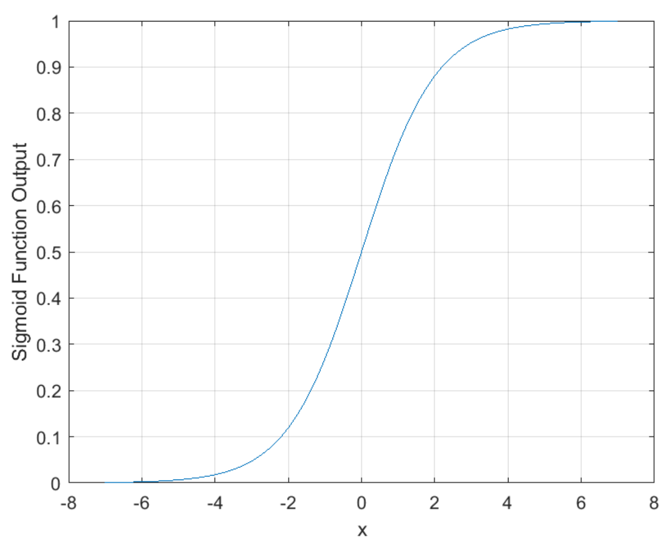

Figure 4.2. *Sigmoid* activation function

The *sigmoid* AF takes the output from Equation 4.1 and normalizes it to a value between 0 and 1. The more negative values are closer to zero, while more positive numbers are closer to one, as shown in Figure 4.2. The non-linear nature of the *sigmoid* function allows neural networks to better fit curved borders for data classification, although the calculation does take more time relative to linear functions.

4.3.2 *ReLU* AF

The alternative AF investigated in our neural networks is the *ReLU* function, given in Equation 4.3 [23], and is shown in Figure 4.3.

$$\begin{aligned} \varphi(x) &= \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \\ &= \max(0, x) \end{aligned} \tag{4.3}$$

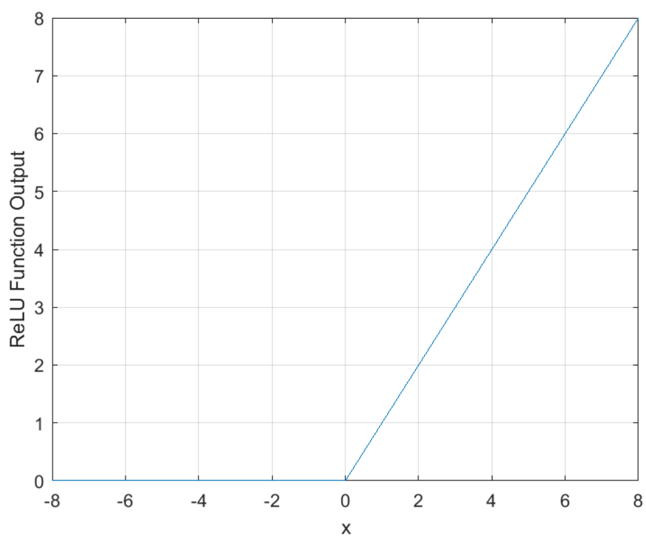

Figure 4.3. *ReLU* activation function

The *ReLU* AF takes the output from Equation 4.1 and sets any negative number equal to zero, and any positive number equal to itself, as shown in Figure 4.3. While the *sigmoid* function limits a HL node's output to unity regardless of its magnitude [23], the *ReLU* function does not limit the positive outputs of a node. This property of the *ReLU* function helps combat the potential for a vanishing gradient in the training process, in which the back-propagation calculations train the later layers, but not the earlier layers [23]. The linear nature of the *ReLU* function allows for faster calculations, though it may not be as powerful in fitting curves to data classification as the non-linear *sigmoid* function.

4.3.3 *Softmax* AF

The *softmax* activation function is particularly useful in calculating the final output for neural networks. The *softmax* function limits the output nodes to a value between zero and one, while also ensuring the sum of all the outputs is equal to one [23]. Equation 4.4 from [23] shows the *softmax* function equation.

$$\begin{aligned} y_i = \varphi(v_i) &= \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3} + \ldots + e^{v_M}} \\ &= \frac{e^{v_i}}{\sum\limits_{k=1}^{M} e^{v_k}} \end{aligned} \tag{4.4}$$

The subscript i in Equation 4.4 denotes the i^{th} node of M total nodes in the current layer, and v_i is the output from that node. Therefore, y_i is the *softmax* output for node i. Because the *softmax* function keeps the sum of all outputs of the neural network equal to one, it is particularly useful for classification when there are more than two classification options.

4.4 Back-propagation for training neural networks

Back-propagation is the method of propagating the output classification error backwards from the output of the neural network to the input in order to train the network [23]. Kim [23] derives the back-propagation algorithm. The final equations utilized in this thesis are given in Equations 4.5 and 4.6 for the training of weights in our neural networks.

$$\begin{aligned} e_i &= d_i - y_i \\ \delta_i &= \varphi'(v_i)e_i \end{aligned} \tag{4.5}$$

Equation 4.5 gives the error, and delta calculations for each node of the neural network. In this equation d_i is the correct output of node i, y_i is the calculated output of node i, and δ_i

is calculated using both the error, e_i, and the derivative of the AF, $\varphi'(v_i)$, utilized in that node [23].

$$\begin{aligned} \Delta W_{ij} &= \alpha \delta_i x_j \\ W_{ij} &\leftarrow W_{ij} + \Delta W_{ij} \end{aligned} \tag{4.6}$$

Equation 4.6 gives the method for calculating the change to each weight, ΔW_{ij}, and then adjusted weight after that change is utilized. In this equation x_j is the input signal for the corresponding weight, W_{ij} [23]. Subscripts i and j denote the i^{th} node in the j^{th} layer of the neural network. The parameter α, utilized in calculating the amount in which to change the neural network weights, is the rate at which the network learns. This rate is a value between zero and one, where higher values of α correspond to more aggressive learning. A network with a higher α might train faster, however it might also never achieve an optimal solution by taking too large of leaps every training iteration. Whereas a lower α might train the network so slowly that the time it takes to achieve an optimal solution is prohibitively long for use in real life applications.

For each input in the neural network's training set, the back propagation algorithm is employed to adjust the weights in and out of each node. This weight adjustment trains the neural network so it can achieve the best P_c when new data is input.

CHAPTER 5:
Neural Network Classification of M-Sequences

For the utilization of a neural network in the classification of m-sequence codes, we assume that the m-sequence received is one of six possible codes, given in Table 3.1. We also assume that a pre-detector (e.g., an energy detector) has already located the signal, i.e., we assume that a signal is present for all inputs. Therefore, the neural network is only utilized in the classification of the m-sequence used to spread the signal.

5.1 Forming the neural networks

Our experiment compares the P_c for three different setups: 1) a *sigmoid* AF with one HL, 2) a *sigmoid* AF with three HLs, and 3) a *ReLU* AF with three HLs.

The variable α is set to 0.9 for the *sigmoid* AF, and set to 0.01 for the *ReLU* AF. The neural network input layer consists of 31 nodes, and each HL also has 31 nodes. The output layer is made up of six nodes; one for each of the six possible m-sequence codes.

As mentioned in Section 4.3, the AFs used in HL outputs are the *sigmoid* and *ReLU* functions, whereas all final output node values are calculated utilizing the *softmax* function. This ensures each output can be used later for our P_c calculation. The largest value from that output calculation is then set as the network classification when testing the trained neural network.

5.2 Forming training and test sets

To create training and test data sets for deep learning, we ensure an equal number of each code throughout the sets. In the training set, this well-balanced data set ensures the neural network does not over-fit classification of one code over another. For the test set, this allows an accurate representation of the neural network's ability to classify signals.

In our data set, we ensure an equal number of each m-sequence randomized throughout the set. For the ***training*** set, there are 100 of each possible code for a total of 600 arrays. For the ***test*** set, there are 1000 of each possible code for a total of 6000 arrays.

5.2.1 Gaussian white noise addition to training set

To investigate the effect noise in the training set has on P_c in the test set, we add randomly generated Gaussian white noise to our training data. We utilize one training set with no noise added, and four others with SNRs of 5 dB, 10 dB, 15 dB, and 20 dB.

$$\mathbf{x} = \mathbf{m} + \mathbf{n} \tag{5.1}$$

Equation 5.1 shows the received signal by adding noise to the m-sequence signal in the training set. In this equation $\mathbf{n}$ is the array of random noise of a given SNR, $\mathbf{m}$ is the m-sequence coded waveform, and $\mathbf{x}$ is the total received signal. Variables $\mathbf{n}$, $\mathbf{m}$, and $\mathbf{x}$ are all vectors of length $N = 31$.

5.3 Comparing activation functions

The three neural networks mentioned in Section 5.1 are each trained via the five training sets discussed in Section 5.2. From each of these trained neural networks, the HL weights are saved for use in testing the network to calculate P_c.

Once a network is trained, Gaussian white noise is added to the test set, with SNR between -20 dB and 20 dB. The P_c is then calculated for an entire test set for each value of SNR. Figure 5.1 shows the percentage for each of the neural network to classify the test data vs. the SNR of that test data.

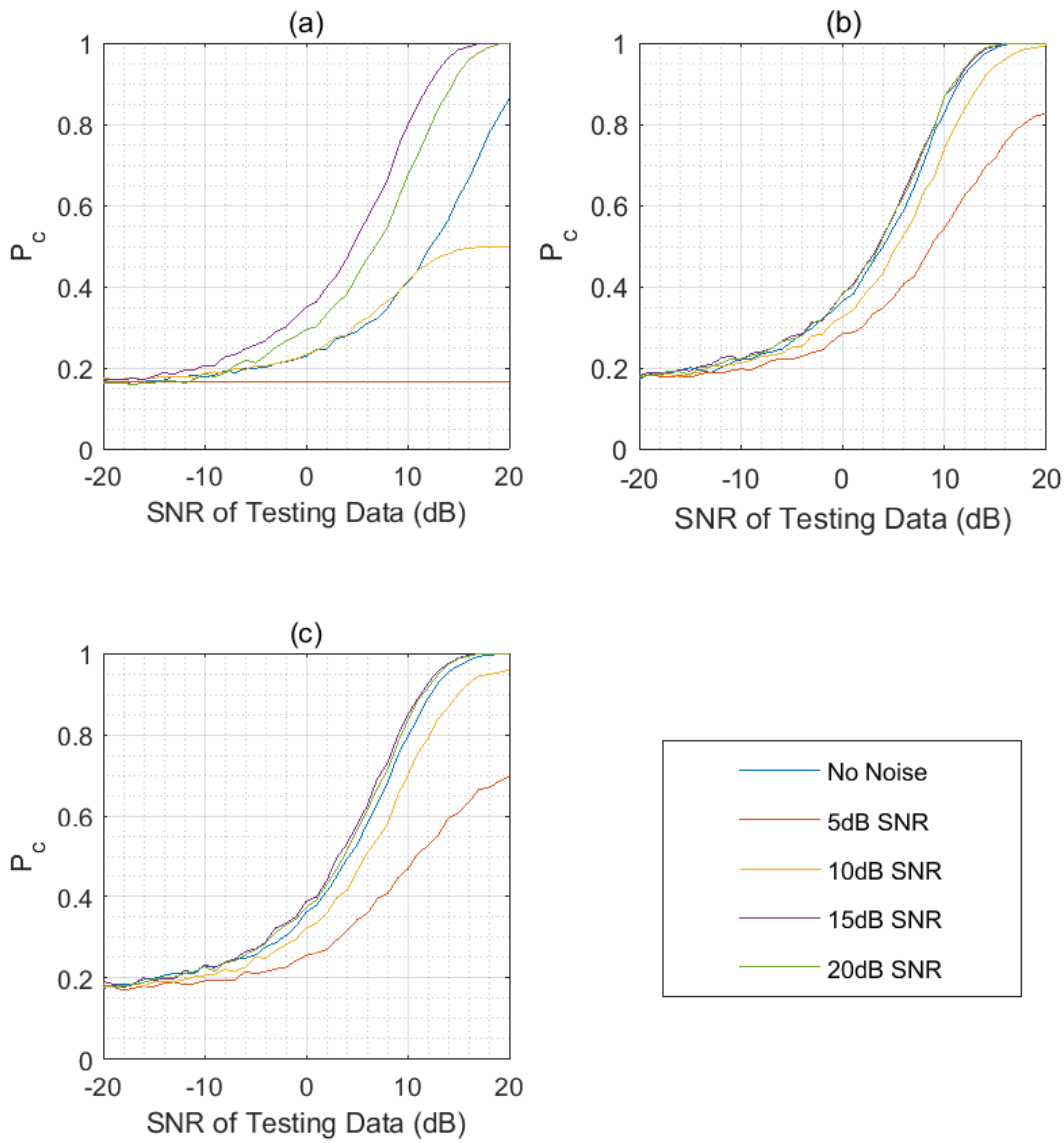

Figure 5.1. Comparison of P_c for various activation functions when random white Gaussian noise added to neural network training set: (a) *ReLU* AF with three HL, (b) *sigmoid* AF with one HL, and (c) *sigmoid* AF with three HL.

The neural network that utilizes the *ReLU* AF with three HL is shown in Figure 5.1(a). Of note on this plot is the line signifying SNR of 5 dB in the training set. We notice when utilizing the *ReLU* AF with a low SNR training set, the trained weights do not converge properly. In the MATLAB simulation, they instead become *not-a-number (NaN)* after just a handful of iterations of training, leading to an inability to classify the signal. In Matlab, *NaN* represents numbers that are neither real nor complex. Through some mathematical calculation in the back-propagation, our weights became either $0/0$ or ∞/∞, resulting in MATLAB labeling the weights as *NaN*.

On all plots in Figure 5.1, an SNR of 15 dB in the training set leads to the highest values for P_c throughout the curves. The next best option for training with the *ReLU* AF is an SNR of 20 dB, whereas the next best when using the *sigmoid* AF is the use of no random noise added to the training set.

5.3.1 Comparing best performing neural networks

The next step in this research is to compare the best performing P_c curve from each network in Figure 5.1 against each other. Figure 5.2 shows this comparison between the best trained network for each neural network, while utilizing the 15 dB SNR training set. The performance comparison shows that, with the neural networks set up as discussed in Section 5.1, the *sigmoid* AF provides better classification of m-sequences of this length.

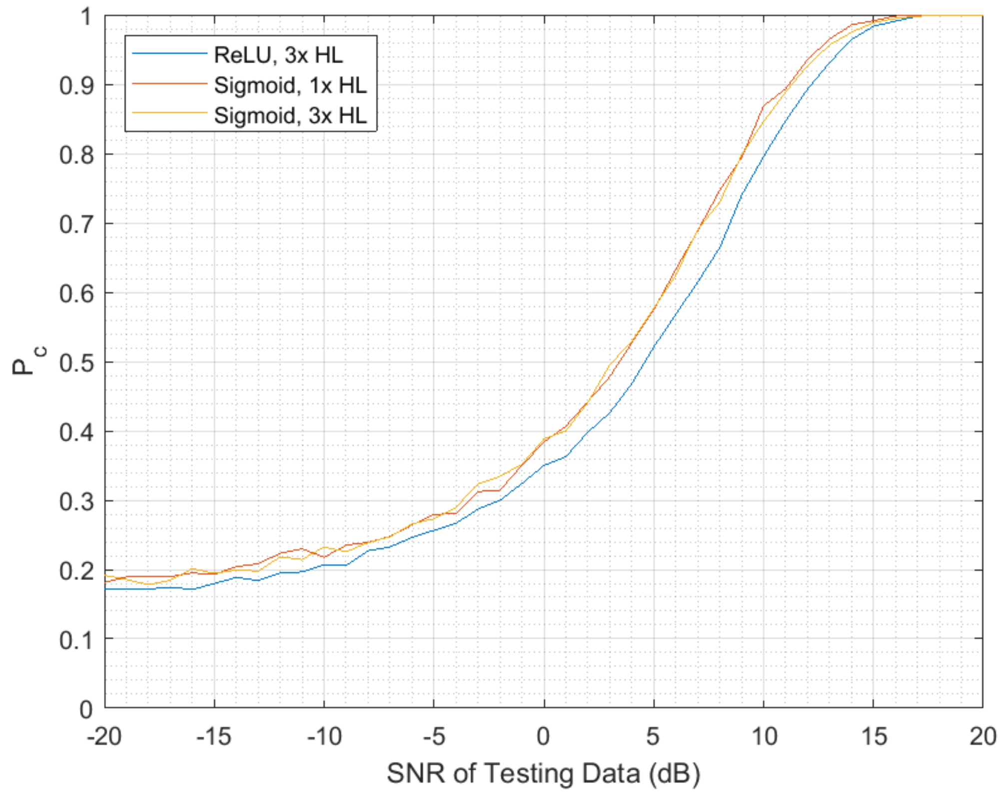

Figure 5.2. Comparison of best P_c line for each activation functions

All neural networks have comparable P_c: at -20 dB SNR, P_c ranges from 17.2% to 19.2%, and at 0 dB SNR, P_c ranges from 35% P_c to 38.9%. Also, the P_c curves for all neural networks converge to over 95% P_c beginning at 15 dB SNR to 100% by 20 dB SNR.

The main difference is the separation between the percentage to classify the m-sequence code at the mid-range SNR test set data, between 0 dB to 15 dB SNR. We see at an SNR of 10 dB, the network trained with the *ReLU* AF and three HL has a P_c = 79.6%, the network trained with the *sigmoid* AF and three HL is at P_c = 84.6%, and the network with *sigmoid* AF and one HL has P_c = 86.9%. There is a 7.3% difference in P_c between use of the *ReLU* AF versus the *sigmoid* AF. It can also be noted that both neural networks trained with the *sigmoid* AF are within 3% P_c throughout Figure 5.2. Therefore, both *sigmoid* networks are comparable in classification performance.

5.3.2 Comparing time to train neural networks

Another comparison of performance for the neural networks is the time required to train the network. Figure 5.3 shows the time to train the three networks. These times are collected on the same computer, on the same day, in which the only program open and running is MATLAB, to ensure a fair time comparison.

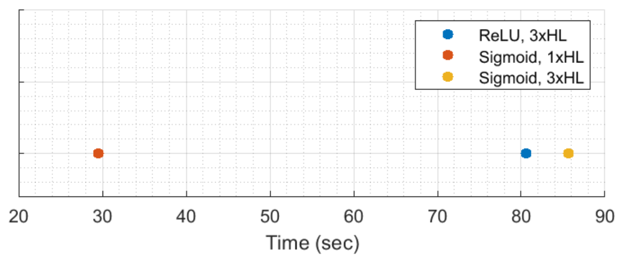

Figure 5.3. Comparison of best computing time required to train the three neural networks.

Figure 5.3 shows that the number of HL plays a large role in the time required to train the network. The network trained with the *sigmoid* AF and one HL completes training in 29.5 s, whereas the networks trained with three HLs take 80.6 s and 85.7 s for the *ReLU* and *sigmoid* trained networks respectively. Of note, both networks with three times the number of HLs increase the time-to-train by almost three times the time it takes to train the network with a single HL.

It is also notable that there is a 5.1 second difference in computation time between the network trained with the *ReLU* AF, a linear function, than with the *sigmoid* AF, a non-linear function. This decrease in time for the *ReLU* function is because the computation for the linear function takes less processing power than for the non-linear function. The difference for a single computation may be insignificant, however when every node uses the function for every iteration of training and back-propagation, the faster speed for the *ReLU* function can make a significant difference.

5.4 Varying the number of nodes in the hidden layers

The neural networks in Section 5.3 are all constructed with 31 nodes in each HL, which is the same number of nodes as the input layer. To investigate if increasing the number of nodes in the HLs can increase the ability of the neural network to classify signals, we double the number of nodes so that each HL now contains 62 nodes each. We then re-train the neural networks utilizing the *sigmoid* AF with a 15 dB SNR training set, for both one and three HLs, to compare performance.

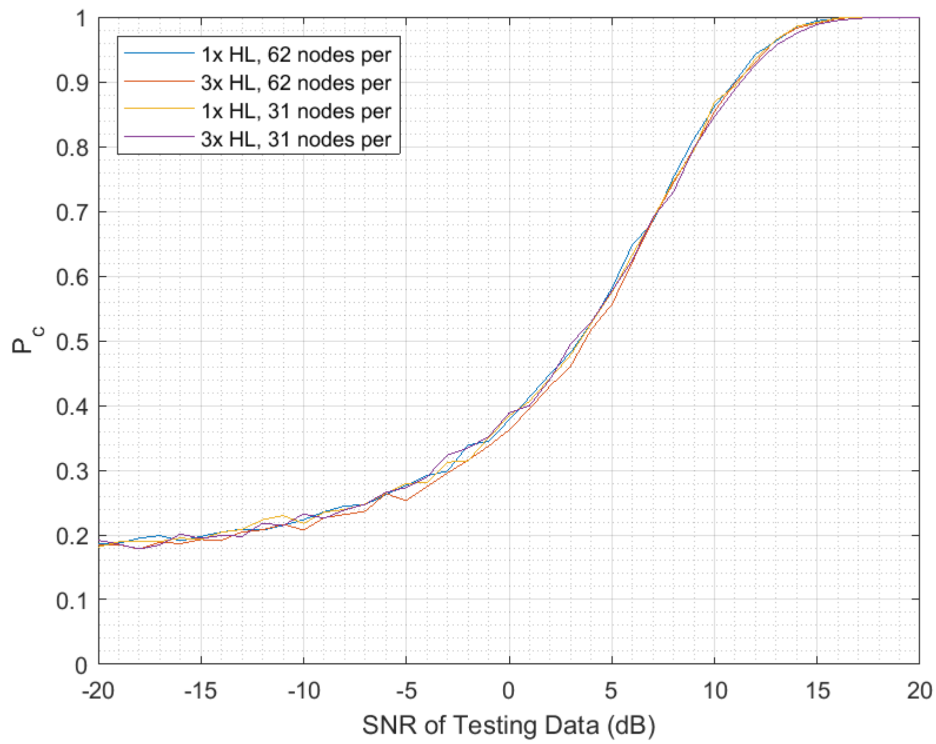

Figure 5.4. Comparison of P_c for neural networks that have 31 nodes in each HL vs. those with 62 nodes in each HL

The outcome of doubling the number of nodes in the HLs shows there is no significant change to the network's P_c curves, as shown in Figure 5.4. The doubling of nodes however doubles the time required to train the networks. The time to train increased from 24.8 s to 57.4 s in the neural network with three HLs. Therefore, the increase in nodes actually worsens the time-to-train performance of the neural network for this application.

5.5 Varying α with the *ReLU* AF

The training rate α utilized in training the *ReLU* neural network is 0.01, a relatively low value compared to $\alpha = 0.9$ utilized in training the *sigmoid* networks. Therefore, we now investigate how an increase in training weight affects classification performance. We utilize the network trained with the *ReLU* AF and both the training set with 15 dB SNR and the training set with no noise added.

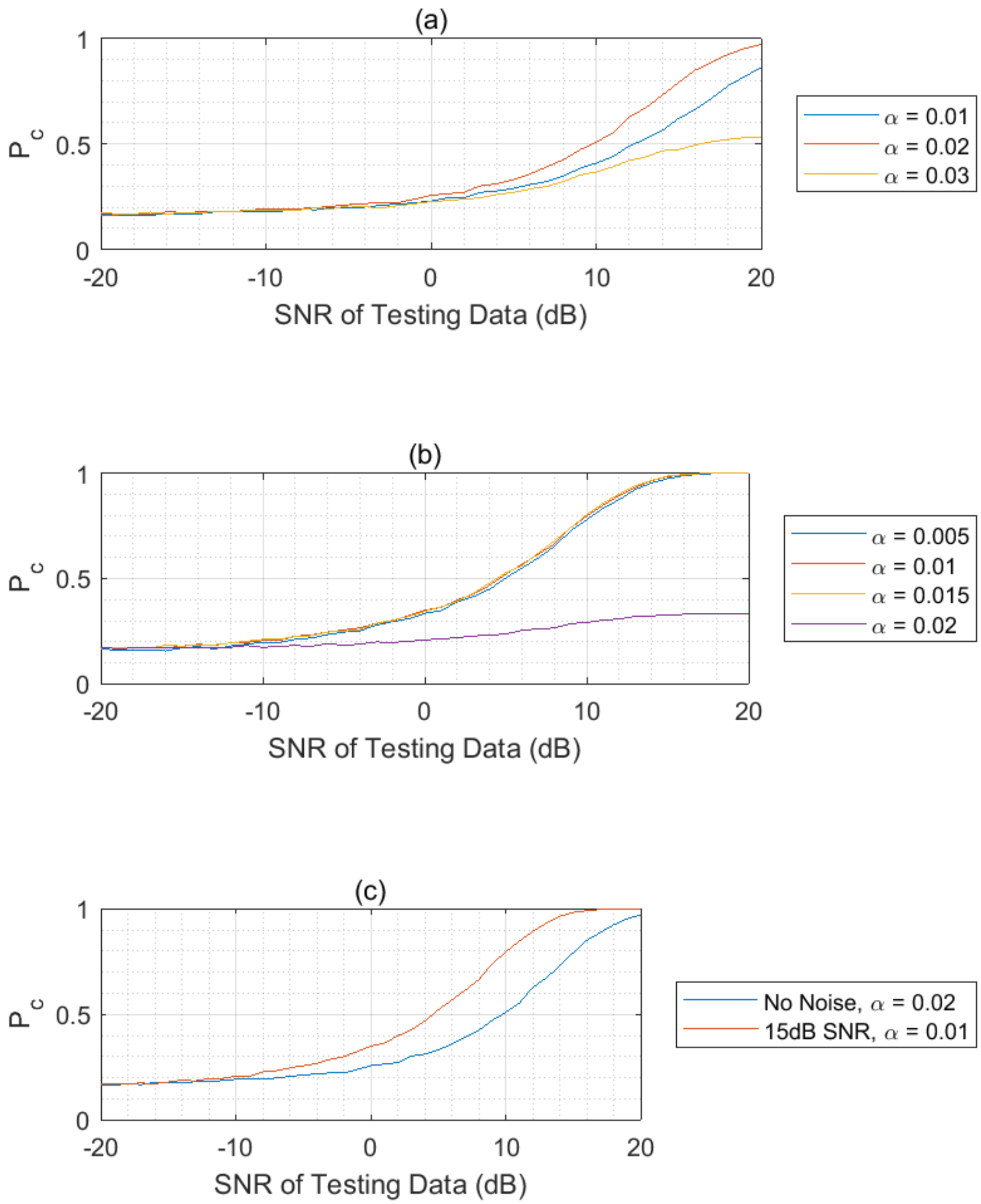

Figure 5.5. Comparison of P_c when varying α utilized in training a neural network with the *ReLU* activation function: (a) no noise added to training set, (b) 15 dB white noise added to training set, and (c) comparison of best α for classification.

The result of increasing training rate with no noise added to the training set is shown in Figure 5.5(a). A slight increase in α to a value of 0.02 increases the ability of the network to classify signals, from Pc = 86.3% to P_c = 97.2%. However, increasing α to a larger rate of 0.03 leads to a decrease in performance with a P_c = 53.2%. This decrease in performance is due to the more aggressive training rate's inability to find the optimal weights, as discussed in Section 4.4.

Calculated P_c for a network trained with a 15 dB SNR training set is shown in Figure 5.5(b). The training rate between 0.005 and 0.015 provides the comparable results, with P_c between 95.3% and 96.5% at as low as 14 dB SNR. Increasing α to 0.02 however, causes a significant decrease in P_c, with a drop to 33.3% at 20 dB SNR.

The comparison of the network trained with no noise training set and $\alpha = 0.02$ to the network trained with a 15 dB SNR training set and $\alpha = 0.01$ is shown in Figure 5.5(c). This plot shows that the neural network trained with a 15 dB SNR training set and $\alpha = 0.01$, which are the same parameters utilized in Section 5.3, maintains the best performance.

5.6 Comparing matched filter bank classification to neural network classification

A more practical performance evaluation of the trained neural networks to classify m-sequences is to compare it to the performance of the traditional matched filter bank method. To that end, the three trained neural networks in this chapter are compared to the matched filter bank from Chapter 3.

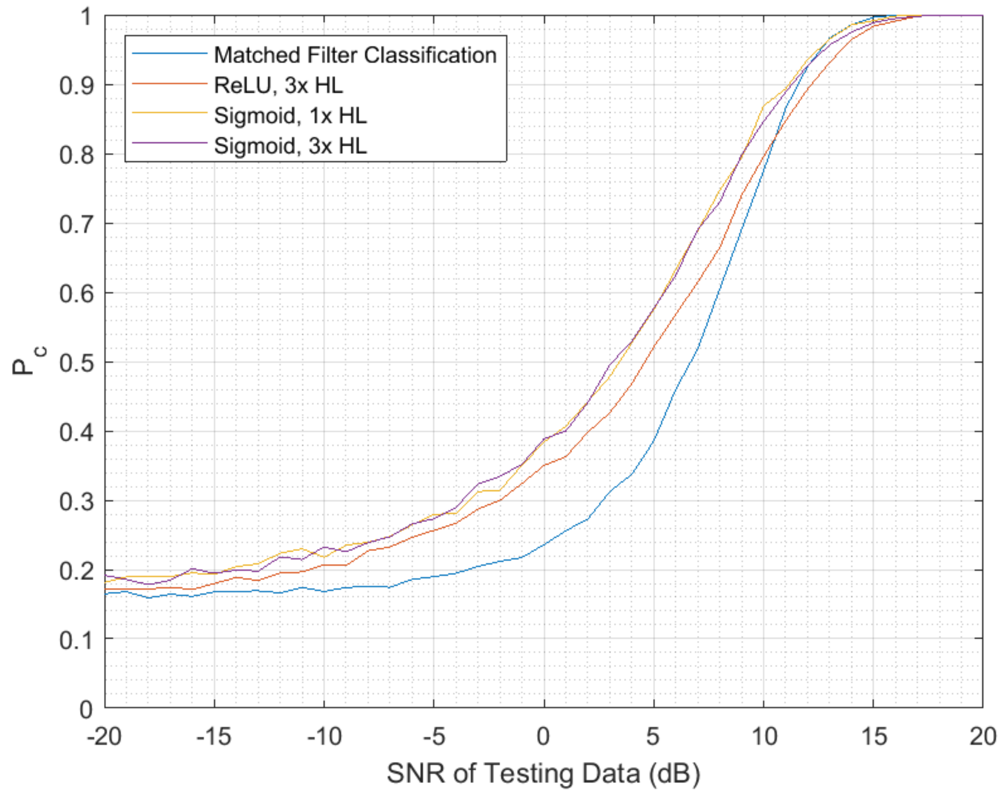

Figure 5.6. Comparison of P_c utilizing traditional matched filter bank method vs. the trained neural networks

The P_c curves for all neural networks trained with a 15 dB SNR training set compared to the P_c for the traditional matched filter are shown in Figure 5.6. At the lowest SNR for testing data, -20 dB, our best performing neural network has P_c = 19.2% and our matched filter simulation has P_c = 16.5%. Also, at the higher SNR, around 15 dB, all the plots begin to converge close to 100% P_c, with the worst performing neural network at P_c = 98.4% and the matched filter at P_c = 99.7%. It is not until between 10 dB to 11 dB SNR that the matched filter begins outperforming our neural networks. The greatest difference in performance between the matched filter and neural networks is seen in the mid-range, SNR between -15 dB and 10 dB when using the testing data.

At 5 dB SNR the neural networks trained with a *sigmoid* AF are performing with P_c =

57.8%, the neural network trained with the *ReLU* AF is performing with P_c = 52.1%, and the matched filter bank method is performing with P_c = 38.5%. This is a 19.3% decrease in P_c when utilizing the traditional method for classification.

5.6.1 Comparison of time to classify

Another performance comparison of the neural networks versus the traditional matched filter bank is the computational time required for the receiver (i.e., computer using MATLAB) to classify the signals. In a tactical environment every second makes a difference in our ability to make timely decisions.

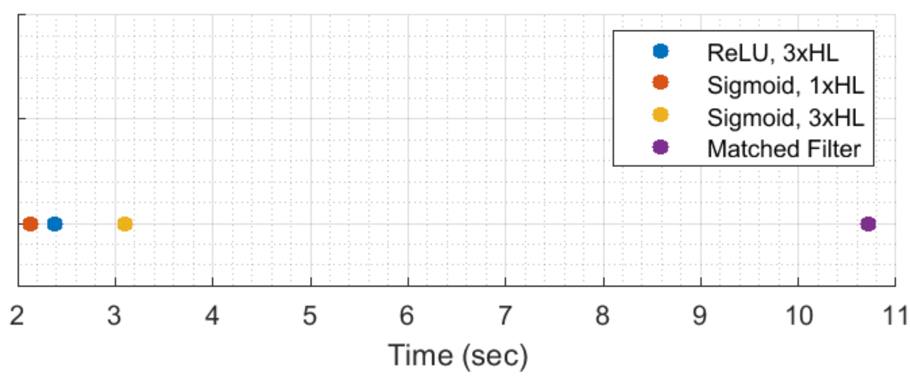

Figure 5.7. Comparison of computing time to classify m-sequences for fully trained neural networks versus a traditional matched filter bank utilizing Monte Carlo simulation.

Figure 5.7 shows the difference in time for fully trained neural networks to classify signals versus the traditional matched filter bank method. The neural network performance in this figure is predicated upon the network already being trained in the classification of m-sequences as discussed previously. There is a 7.6 second decrease in time to classify between the matched filter simulation time of 10.7 s and the *sigmoid* trained neural network with three HL, which classified the test set in 3.1 s. The matched filter bank method takes nearly four times longer to classify than a trained neural network.

For the neural networks, the majority of time is taken up in the time to train the network, as illustrated by the time difference between Figures 5.3 and 5.7. However, in a tactical environment, we would not be training the network. We will be employing our forces with fully trained systems ready to execute.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Neural Network Classification to Include "Noise-Only" Signal

The assumption in Chapter 5 is that the spread signal being classified is present amidst the background noise. An interesting research investigation is to add a "noise-only" classification hypothesis in which the neural network is trained to determine if a signal is present. This "noise-only" alternative bypasses the need for a signal energy detector in our circuit prior to classification. A neural network that is able to accurately determine if a signal is present (or not) without the pre-detection circuit and to classify that signal, if present, may increase decision making capabilities by decreasing receiver processing time (since the signal energy detector is removed).

Throughout this chapter a neural network trained to determine if an m-sequence coded signal is present or not will be referred to as having been trained with a "noise-only" classification option. This null hypothesis allows these networks to have seven possible outcomes. On the other hand, neural networks trained as discussed in Chapter 5, in which the signal is assumed present, will be referred to as having been trained without the "noise-only" hypothesis.

6.1 Probability of classification with and without "noise-only" hypothesis

Comparison of the neural networks ability to classify signals with the addition of a "noise-only" outcome versus neural networks without such an option allows us to check the tactical viability of this training. The addition of the "noise-only" hypothesis when random white noise is added to the training set causes the network to falsely train node weights to classify the noise as a signal and vice versa depending on the SNR utilized for the training set as may be expected. Our ***training*** set consists of 100 of each possible classifications, the same as in Chapter 5, for a total now of 700 arrays. Our ***test*** set consists of 1000 of each possible classifications, for a total of 7000 arrays.

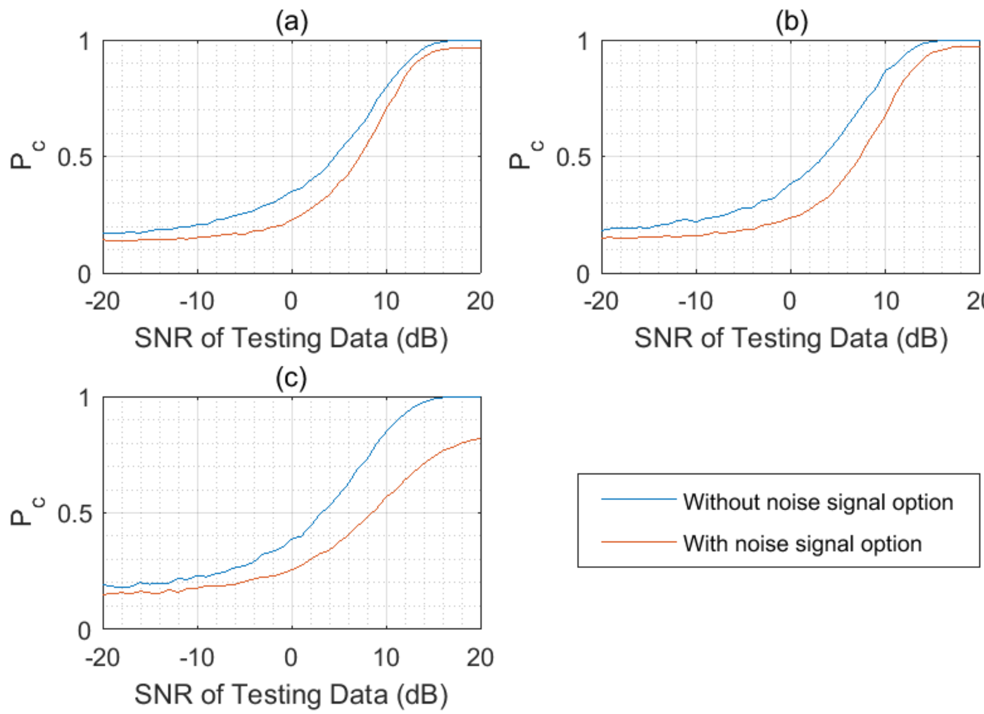

Figure 6.1. Comparison of P_c for neural networks trained with and without "noise-only" classification options: (a) *ReLU* AF with three HL, (b) *sigmoid* AF with one HL, and (c) *sigmoid* AF with three HL.

Because the 15 dB SNR training set afforded the best outcomes in Section 5.3, we will use the same SNR in training the networks with "noise-only" classification options. Figure 6.1 shows P_c curves for each network for training with versus without the "noise-only" hypothesis. Figure 6.1(a) shows the *ReLU* AF with three HL, (b) shows the *sigmoid* AF with one HL, and (c) shows the *sigmoid* AF with three HL.

For all neural networks, the training "without the noise-only" hypothesis outperforms that with the additional classification category (i.e., the null alternative). At 10 dB SNR, the *ReLU* trained network with three HL shows a 9.2% difference in P_c, the *sigmoid* trained network with one HL a 19.3% difference, and with three HL a 27.8% difference between the training sets with and without the "noise-only" classification option.

The networks trained with a "noise-only" alternative never achieve 100% P_c, where each curve levels off below 100% accuracy. In other words, there will always be some probability that one of the hypotheses is misclassified including the null alternative being classified as a signal. At 20 dB SNR, the network trained with the *ReLU* AF levels off at P_c = 96.7%, the network trained with the *sigmoid* AF and one HL levels off at 97.2%, and three HL at 82.0%. This lack of 100% P_c may be tactically acceptable depending on the specifications set by the missions in which the classifier is employed in.

6.2 Probability of false alarm (P_{fa}) with "noise-only" classification option (or null alternative)

Another risk associated with excluding the energy detector from our circuit is that the neural network may falsely deem that an m-sequence coded signal is received when there is in fact no signal, i.e., only noise is present. The P_{fa} in energy detectors is well discussed in the literature and that P_{fa} depends on two factors: noise energy present and some threshold based on expected signal energy if signal were present.

To test the P_{fa} of the neural networks, we will add a test set with only noise. This test set's noise varies in power from a variance, σ^2, between 0 (which corresponds to no noise) to 0.2. The P_{fa} is then calculated for each value of σ^2 and plotted.

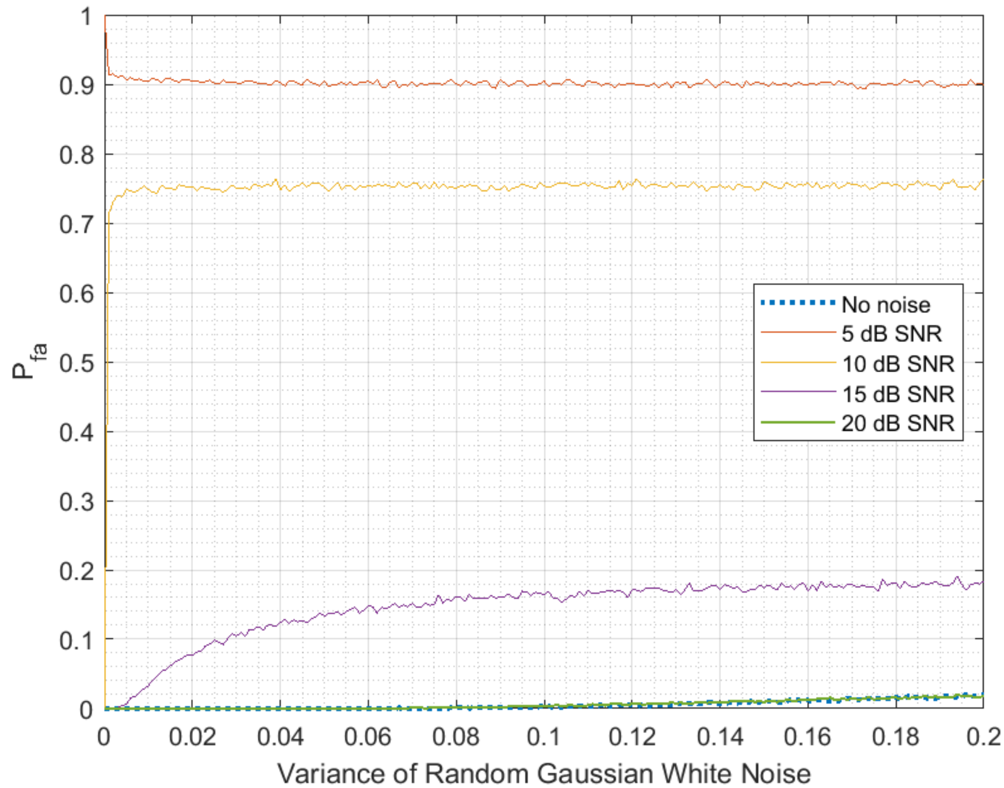

Figure 6.2. Probability of false alarm for a neural network trained with the *sigmoid* AF and one HL that contains a "noise-only" classification option

Figure 6.2 shows that, for the *sigmoid* AF with one HL, if the neural network is trained with 5 dB or 10 dB SNR in the training set, there is greater than 70% P_{fa} for σ^2 as low as 0.001. For the network trained with a 15dB SNR training set that percentage drops to 18% P_{fa} for σ^2 greater than 0.16. Recall that classification performance of networks trained with a 15 dB training set is decided to be the best among the ones tested. If 18% P_{fa} is an acceptable margin for non-stringent systems, then a neural network of this type is an acceptable option.

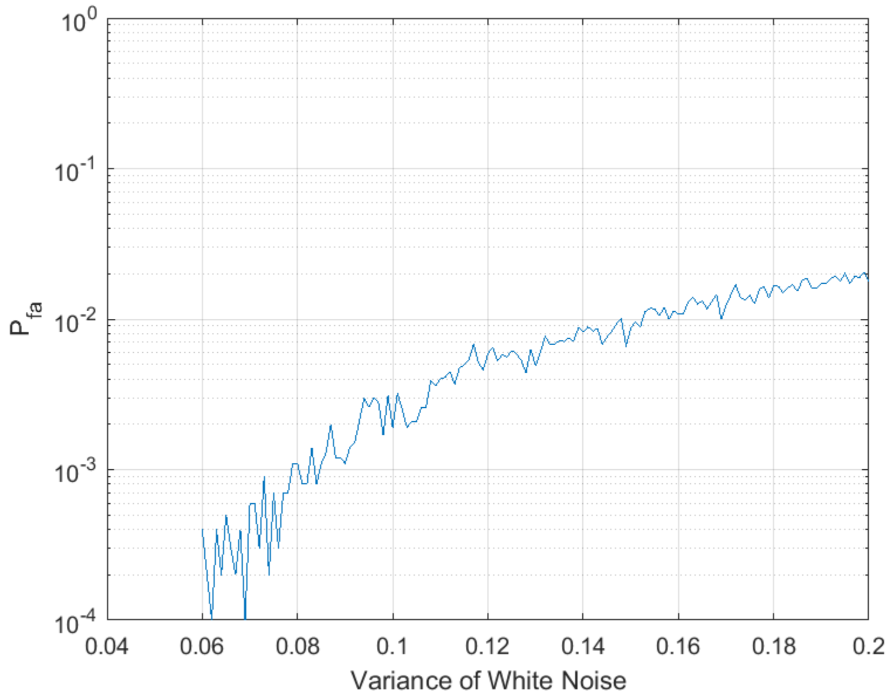

Figure 6.3. Probability of false alarm for neural network with 1 HL, using the *sigmoid* AF, and trained with no noise added to the training set.

When the network is trained with either no-noise or a 20 dB SNR (i.e., very low noise power) training set, the P_{fa} is greatly reduced. This is because in these instances the neural network has a better idea as to what a "no noise" signal should look like because it is trained with low noise. Figure 6.3 shows a semi-log plot of the *sigmoid* network trained with no noise added to the training set. It shows that for the higher noise variance, the P_{fa} is still around 1%, a much more desirable false alarm rate than the 18% seen in Figure 6.2.s

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7: Conclusion

This study has shown that the use of trained neural networks as a classifier of m-sequence coded signals offers improved classification performance at mid-range SNR values compared to the classical matched filter bank approach. We determined that a neural network trained with the *sigmoid* AF, one HL, and in which the training set is set to 15 dB SNR offered the best classification performance compared to other variations in this work. This network offered the best P_c curve with the lowest time required for both training the network and testing the classifier. We also determined that the neural network with 31 nodes per HL offered the same performance as one with 62 nodes per HL, while the lower number of nodes decreased the time to train by half.

We also investigated the option of utilizing the neural network to not only as a classifier where spread signal is assumed present, but in a case where the "noise-only" signal is a valid hypothesis and thus an alternative classification outcome. There is a noticeable decrease in performance, as the neural network converged at a P_c less than 100%. And, if tactically acceptable, given a training set of higher SNR, the P_{fa} was relatively low, which attests to neural network's potential viability as an option.

7.1 Future work

Though the neural network trained and tested with m-sequence codes that are 31 chips in length showed an increased performance over matched filter bank classification, there is much more to explore. These networks were only tested on a handful of the possible m-sequences. Future research could expand the neural network to look at m-sequences of various lengths instead of a set number of chips. Also, investigating different spread spectrum sequences will be a good study.

These networks were also created via MATLAB code, which is not a programming language utilized in hardware. Further future work could include the training of a neural network in software languages such as C++ or PYTHON and even a true hardware language like Verilog. This network could then be placed into a circuit and tested for the feasibility of actual use in tactical scenarios.

List of References

[1] R. E. Blahut, *Modem Theory: An Introduction to Telecommunications*. Cambridge: Cambridge University Press, 2009.

[2] R. E. Ziemer, *Fundamentals of Spread Spectrum Modulation*, 1st ed. (Synthesis Lectures on Communications). San Rafael, Calif (1537 Fourth Street, San Rafael, CA 94901 USA): Morgan Claypool Publishers, 2007, vol. 3., no. 1.

[3] J. Tou, P. Yip, and H. Leung, "Spread-Spectrum Signals and the Chaotic Logistic Map," *Circuits, systems, and signal processing*, vol. 18, no. 1, pp. 59–73, 1999.

[4] L.-L. Yang and L. Hanzo, "Acquisition of M-Sequences using Recursive Soft Sequential Estimation," *IEEE transactions on communications*, vol. 52, no. 2, pp. 199–204, 2004.

[5] X. J. Li, Z. C. He, and Q. Li, "Research of Direct Sequence Spread Spectrum Communication System Based on MATLAB," *Applied Mechanics and Materials*, vol. 103, pp. 279–284, 2011.

[6] C. Li, W. Bao, L. Xu, H. Zhang, and Z. Huang, "Radar Communication Integrated Waveform Design Based on OFDM and Circular Shift Sequence," *Mathematical problems in engineering*, vol. 2017, pp. 1–10, 2017.

[7] M. C. Jeruchim, "Modeling and Simulation of Communication Systems: an Overview," *Journal of the Franklin Institute*, vol. 332, no. 5, pp. 521–533, 1995.

[8] R. L. Peterson, R. E. Ziemer, and D. E. Borth, *Introduction to Spread Spectrum Communications*. Upper Saddle River, NJ: Prentice-Hall, Inc., 1995.

[9] B. Sklar, *Digital Communications : Fundamentals and Applications*, third edition. ed. London: Pearson, 2021.

[10] K. Fyhn, T. Arildsen, T. Larsen, and S. H. Jensen, "Demodulating Subsampled Direct Sequence Spread Spectrum Signals using Compressive Signal Processing," in *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE, 2012, pp. 2556–2560.

[11] D. C. ARAUJO, G. FERRE, C. C. CAVALCANTE, and I. M. GUERREIRO, "A Spectral Efficiency Enhancement for Chirp Spread Spectrum Downlink Communications," in *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, 2020, pp. 1–6.

[12] K. Takahashi, K. Inoue, Y. Kawakita, J. Mitsugi, and H. Ichikawa, "A Method to Assign Spread Codes Based on Passive RFID Communication for Energy Harvesting Wireless Sensors Using Spread Spectrum Transmission," *EAI Endorsed Transactions on Ambient Systems*, vol. 2, no. 6, pp. 242–4, 2015.

[13] S. Lei and Z. zhijing, "Blind Estimation of the Pseudo-random Sequences of Direct Sequence Spread Spectrum Signals in Multi-Path Using Fast ICA," in *2009 Pacific-Asia Conference on Circuits, Communications and Systems*. IEEE, 2009, pp. 531–535.

[14] G. Kaddoum, F. Richardson, and F. Gagnon, "Design and Analysis of a Multi-Carrier Differential Chaos Shift Keying Communication System," *IEEE transactions on communications*, vol. 61, no. 8, pp. 3281–3291, 2013.

[15] M. A. Richards, J. A. Scheer, and W. A. Holm, *Principles of Modern Radar (Vol. I: Basic Principles).* SciTech Publishing, 2010, vol. 1.

[16] R. Y. Rubinstein, *Simulation and the Monte Carlo Method*, third edition. ed. (Wiley Series in Probability and Statistics). Hoboken, New Jersey: Wiley, 2017.

[17] T. C. Clancy, A. Khawar, and T. R. Newman, "Robust Signal Classification Using Unsupervised Learning," *IEEE transactions on wireless communications*, vol. 10, no. 4, pp. 1289–1299, 2011.

[18] C.-Y. Hung, B. C. Jiang, and C.-C. Wang, "Evaluating Machine Learning Classification Using Sorted Missing Percentage Technique Based on Missing Data," *Applied sciences*, vol. 10, no. 14, pp. 4920–, 2020.

[19] H. Elyousseph and M. L. Altamimi, "Deep Learning Radio Frequency Signal Classification with Hybrid Images," in *2021 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. IEEE, 2021, pp. 7–11.

[20] L. Huang, W. Pan, Y. Zhang, L. Qian, N. Gao, and Y. Wu, "Data Augmentation for Deep Learning-Based Radio Modulation Classification," *IEEE access*, vol. 8, pp. 1498–1506, 2020.

[21] C. Weber, M. Peter, and T. Felhauer, "Automatic Modulation Classification Technique for Radio Monitoring," *Electronics letters*, vol. 51, no. 10, pp. 794–796, 2015.

[22] G. Ramaswamy, "Adaptive Classification of Interfering Signals in a Shared Radio Frequency Environment," in *[1992 Proceedings] The Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 1992, pp. 572–576.

[23] P. Kim, *MATLAB Deep Learning : with Machine Learning, Neural Networks and Artificial Intelligence*. New York, NY: Apress, 2017 - 2017.

[24] U. K. Majumder, *Deep Learning for Radar and Communications Automatic Target*. Norwood, Massachusetts: Artech House, 2020 - 2020.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California