Communication Systems Project

ECE 132A Communication Systems – Winter 2019

University of California, Los Angeles

Professor Suhas Diggavi

Team Members:

Andrew Braun, UID: 304809464

Yaniv Tismansky, UID: 104902137

Zachary Kum, UID: 904736097

About the Team

Andrew Braun:

Andrew is currently a third year at UCLA pursuing a B.S. in Electrical Engineering and will graduate in 2020. On campus he is involved as a technical lead on ELFIN which is a NASA sponsored program in the Engineering School. This summer he will be interning at The Aerospace Corporation as a communication systems engineering intern. In his spare time, he enjoys taking hikes around the LA area.

Yaniv Tismansky:

Yaniv is currently a fourth year at UCLA finishing up his B.S. in Electrical Engineering and will graduate in 2019. On campus he is involved with UCLA Transfer Student Organizations. After graduating he will be working at Northrop Grumman as a Software Engineer in their Mission Systems Sector. In his spare time, you can find him hiking and going to the gym.

Zachary Kum:

Zachary is currently a third year at UCLA working towards a B.S. in Electrical Engineering and will graduate in 2020. On campus he is involved as an Engineering Ambassador where he gives campus tours and works on recruiting prospective undergraduate engineering students. This summer he will be interning at Northrop Grumman as a Systems Engineer in the company's Mission Systems Sector. In his spare time, he likes to play tennis and also works as a tennis coach for UCLA Recreation.

Table of Contents

Introduction

Communication systems are an essential part of modern society and the technology has allowed the word to become greatly interconnected. The purpose of this project is to demonstrate the viability and usefulness of deep learning in wireless communications. In traditional communication systems, a generated message of *k* information bits is encoded to output codes of block size *N*. Standard modulation schemes such as BPSK, QPSK, and QAM would then help transmit these encoded messages across the wireless channel where they would be demodulated and decoded at the receiver. A standard system following this architecture can be seen below in Figure 1.

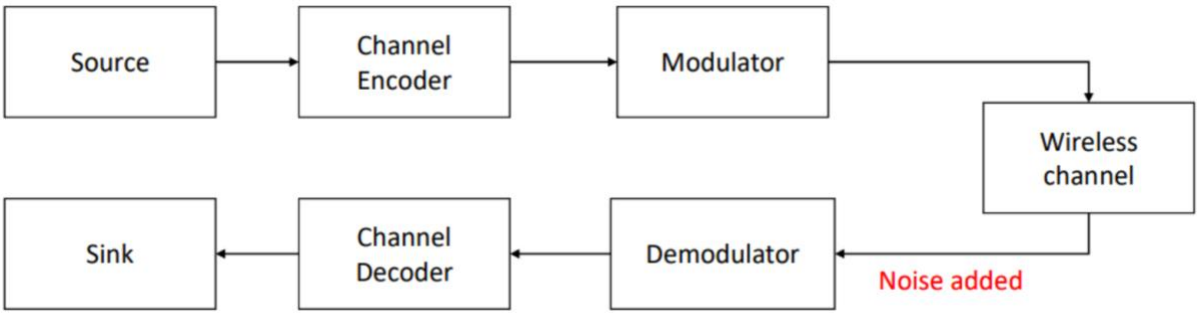

Figure 1. Block diagram of typical communication system.

Utilizing the knowledge of the components necessary to create a complete communication system, the goal was to replace the demodulation and decoding process at the receiver end with a neural network decoder that would use artificial intelligence. A decoder with artificial intelligence capability would be able to be trained in order to decode in one step while also being blind to the model of the channel used. With a robust artificial intelligence algorithm in place, this kind of smart receiver could be used for different coding schemes that it may not be hard programmed for. Using the Python coding language, our team worked to design and test this neural network decoder on simulated channels.

System Architecture

A basic communication system can be broken up into 3 distinct modules which include a transceiver, channel, and receiver. The transceiver includes the channel encoder and modulator, the channel introduces noise which may corrupt the transmission, and the receiver includes the demodulator and channel decoder. The source is the message that will be sent and the sink is the destination for that message. While the artificial intelligence enabled decoder is the primary objective of this project, the entire communication system is simulated. A traditional communication system seen in Figure 1 was simulated which included a channel encoder, modulator, wireless channel, demodulator, channel decoder and sink. This would allow for baseline data to be collected of the received messages which could further be used to train the neural network. The artificial intelligence enabled decoder would replace the traditional demodulator and channel decoder. This new architecture is seen below in Figure 2.

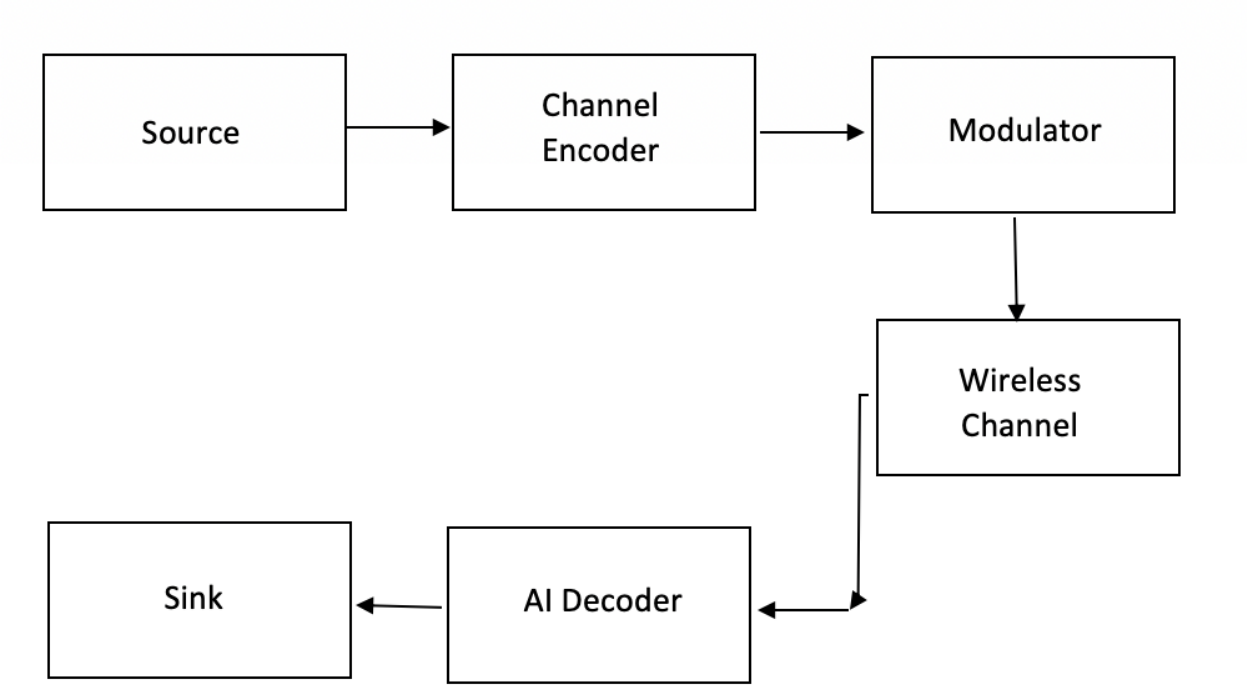

Figure 2. Wireless communication system block diagram with artificial intelligence enabled decoder.

The artificial intelligence chosen for this smart decoder is a neural network. The implementation of a complete conventional communication system was essential for the design of our neural network. Machine learning can be a powerful tool to create systems that are more modular and

less static than current designs. The neural network implemented is form of machine learning where different machine learning algorithms work together to process complex data. The goal is that the neural network would be able to learn to perform calculations it may not be hard coded to compute by comparing to previous examples. Our neural network design required data from the fully simulated conventional system in order to train the system to make the correct decision by comparing the inputted data with previous samples.

The neural network implemented in this communication system is divided into three different layers which comprise of the input layer, the hidden layer, and the output layer. The input layer is simply the message that the neural network receives from the simulated wireless channel. The output layer is the summation of the weights multiplied with the inputs plus some bias which can be seen in below in Figure 3.

$$Y = \sum (weight * input) + bias$$

Figure 3. Equation describing the output of the neural network.

The weight and bias values of this equation are essential for the functionality of the neural network and are computed in the hidden layer. The hidden layer is the heart of any neural network. Here, different neurons are interconnected to calculate the weights and biases given the inputs provided to the system. The neural network is iterated through multiple times in order to update the weight and bias values which contributes to decreased output errors. The architecture of the workings of the hidden layer can be seen below in Figure 4.

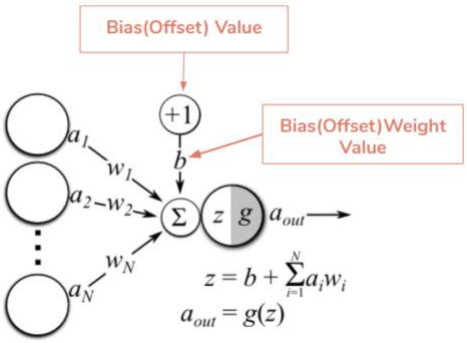

Figure 4. Diagram of the hidden layer of a sample neural network.

In this project's neural network, it was trained so that based on a given input from the channel, it could decide based on a best match principle, which modulation scheme was sent from the transceiver. In order for the neural network to make this decision, a fully functional back end communication system was simulated. Using a series of messages modulated using the binary

phase shift key (BPSK), quadrature phase shift key (QPSK), and quadrature amplitude modulation (QAM). From that simulated system, the probability of error, P_e, was calculated for each modulation scheme. This could then be used in the neural network as a series of cases. When the messages from the transceiver goes through the noisy channel and reaches the neural network capable decoder, it can try the different demodulation schemes for BPSK, QPSK, QAM and compare which one would yield the lowest probability of error. Anything prior to the smart decoder is like a black box to it so any modulation can be used. Utilizing this architecture, the communication system would be able to correctly output the message with minimal errors compared to the transceiver signal.

System Performance Metrics

A communication system can be evaluated using a few important performance metrics. Probability of error is important as it allows the neural network to decide which modulation scheme was most probable at the time of modulation in the transceiver. The maximum probability of signal error for the three modulation schemes BPSK, QPSK, and QAM that can be utilized for this system are as follows:

1. Binary Phase Shift Key Modulation (BPSK) Maximum Probability of Error

Signal to Noise Ratio = SNR = $\frac{E}{\sigma^2}$

$$P_{e|H=i} = Q\left(\frac{d_{min}}{2\sigma}\right) for\ i = 0,1\ [Digital\ Bits] \quad (1)$$

$$P_e = P_{e|H=0} + P_{e|H=1} = 2Q(\frac{d_{\text{minimum}}}{2\sigma})$$

The constellation points for [-1 and 1] are spaced out by $d_{minimum} = 2\text{Eb}$

$$P_e = 2Q(\frac{2}{\sqrt{2N_o}})$$

2. Quadrature Phase Shift Key Modulation (QPSK) Maximum Probability of Error

Signal to Noise Ratio = SNR = $\frac{E}{\sigma^2}$

$$P_e = 2Q(\frac{\sqrt{2E_b}}{\sqrt{N_o}})$$

3. Quadrature Amplitude Modulation (QAM) Maximum Probability of Error

Signal to Noise Ratio = SNR = $\frac{E}{\sigma^2}$

$$P_e \leq (M-1)Q\left(\frac{d_{min}}{2\sigma}\right) \quad (1)$$

$$\text{Let } \frac{d_{min}}{2\sigma} = \sqrt{\frac{E}{\sigma^2}\frac{3}{M-1}} \quad (2)$$

$$= \sqrt{SNR * \frac{3}{M-1}}$$

Substituting (2) into (1),

$$P_e \leq (M-1)Q\left(\sqrt{SNR * \frac{3}{M-1}}\right)$$

For M = 4 possible messages, this reduces to:

$$P_e \leq 3Q\left(\sqrt{SNR}\right)$$

In addition to using probability of error, actual system performance can be quantified by bit error rate (BER) and normalized validation error (NVE). Bit error rate is the rate at which there are errors in individual bits in the transmission of the signal. There is another related error rate called message error rate (MER) where it quantifies the error rate for the entire message. Message error rate is stricter than bit message rate but because our system is merely using simulated messages with little impact to critical infrastructure, bit error rate is appropriate to use. Bit error rate can be computed as a function of signal to noise ratio (SNR) utilizing both training and validation data. This can be represented by the following equation:

4. Calculating Bit Error Rate (BER) as a Function of SNR for Training and Validation Data:

Let Signal to Noise Ratio = SNR = $\rho_t \; or \; \rho_v$

Where ρ_t is SNR of training data and ρ_v is SNR of validation data

Bit Error Rate = BER = $BER_{NN}(\rho_t, \rho_v)$

Normalized validation error is also a function of signal to noise ratio (SNR) for training data and validation data. As NVE goes to 1, the decoder performs similar to a maximum a posteriori decoder which would indicate that our neural network capable decoder has achieved a satisfactory level of performance. NVE can be calculated using the following equation:

5. Calculating Normalized Validation Error (NVE) as a Function of SNR for Training and Validation Data:

Let Signal to Noise Ratio = SNR = $\rho_t \; or \; \rho_v$

Where ρ_t is SNR of training data and ρ_v is SNR of validation data

Normalized Error Rate = $NVE(\rho_t) = \frac{1}{S}\sum_{s=1}^{S} \frac{BER_{NN}(\rho_t, \rho_{v,s})}{BER_{MAP}(\rho_{v,s})}$

Using the performance metrics of probability of error, bit error rate, and normalized validation error will allow for an in depth and thorough validation for the neural network trained decoder in the communication system.

Design Methodology

A conventional communication system typically consists of a message source, channel encoder, modulator, wireless channel, demodulator, channel decoder, and a message sink which has the final transmitted message. Our project was coded in the python language because of its modularity and ability to easily implement TensorFlow for the neural network part of the project, and for its easy-to-use API and integration with other software (as compared to MATLAB). In order to go over our design methodology, we will discuss how we implemented each part of our model and any issues faced during development, what tests were done to ensure correct functionality of our Neural Net decoder (and our other subsystems implemented), and finally any improvements that could have been made.

The message source consists of any sort of text files that will be used as the messages to be sent through the simulated communication system. When first testing our team used too many messages to where the data ended up being multiple gigabytes which was impractical to test with using standard consumer grade laptops. However, once the number of messages was reduced to a manageable number, performance of the system increased and simulated processing time dropped by the minutes. The channel encoder's function was to take the raw binary data from the message and encode it utilizing one of three chosen schemes into another set of binary data. The chosen encoding schemes were Hamming (7,4) Codes, Low Density Parity Check Codes (LDPC), and Random Codes and were chosen because these encoding schemes were the most familiar to the team during development. The next module that was implemented was the modulator class that was responsible for serving encoded data to a modulated QPSK, BPSK, or QAM carrier signal. In respect to each quadrant in the unit circle (Quadrant 1, 2, 3, 4), the QPSK and QAM implementation utilized four different codes in the order of 00, 01, 11, 10.

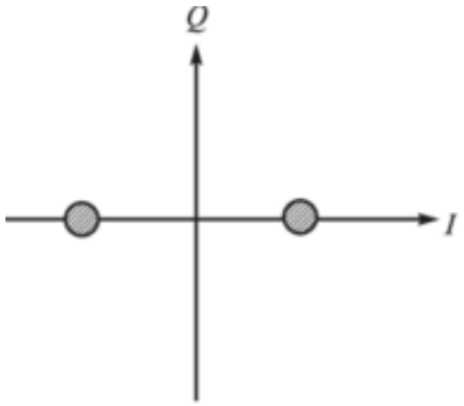

Figure 5. Constellation diagram for BPSK.

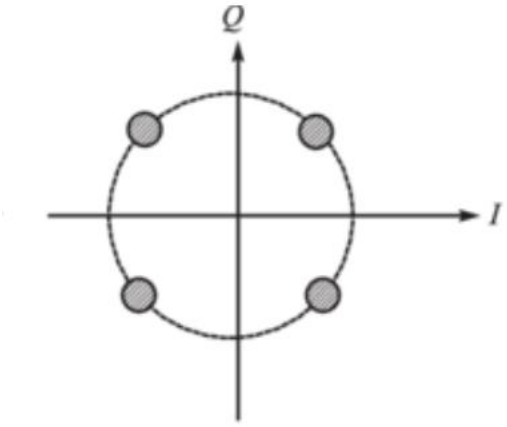

Figure 6. Constellation diagram for QPSK.

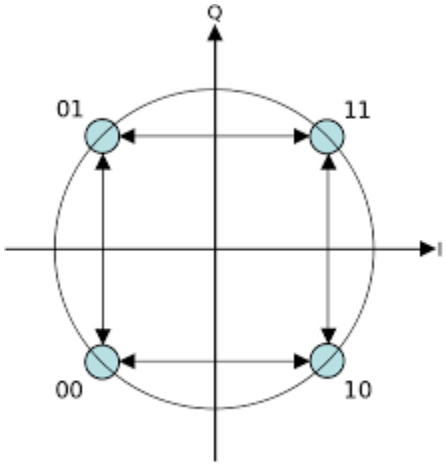

Figure 7. Constellation diagram for QAM.

When the modulation schemes were under development, a problem did arise when the QPSK and QAM schemes failed to demodulate, but this was corrected when they were redesigned to use

complex samples instead of purely real samples. The wireless channel was relatively simple to implement, as it consisted only of the outputted signal from the modulator with the addition of additive white gaussian noise (AWGN) in order to simulate a wireless communication channel. In reality, OFDM should have also been implemented but we did not have enough time to implement and test it for our purposes. Once the signal was sent through our AWGN Channel, it was sent to the demodulator. The demodulator was implemented by using correlators on the input signal (from the AWGN) and multiplied with their respective orthonormal bases to recover the original bits. For QPSK and QAM, this meant multiplexing and demultiplexing for the carrier signal. Additionally, our team did experience an issue in the QPSK and QAM implementations where multiplexing an odd length bit stream resulted in errors, but this was fixed by introducing zero padding to make the I and Q parts of the signal equal length.

Afterwards, the channel decoder then takes the demodulated signal and decodes for the Hamming (7,4), LDPC, and Random Coding schemes. Once decoded, the sink in our system simply outputs the received message into a ".txt" file for further processing.

The implementation of a complete conventional communication system was essential for the design of our neural network. The neural network in this wireless communication system is a traditional neural network where waveform samples from the decoder are used to create trained input data where the neural network could output data as close as possible to the data sent from the encoder. The weights and biases discussed in the System Architecture section are calculated using the python TensorFlow function *tf.random_normal()*. This function would return a tensor which is a matrix of specified dimension containing values that are calculated using a gaussian distribution with specifications $\eta \sim (0, \sigma^2)$. The outputs are then calculated using an activation function. An activation function is a function that allows the neural network to be nonlinear which changes the outputs to take on values between 0 to 1. The activation function used in the neural network is a sigmoid function shown below in Figure 8.

$$S(x) = \frac{1}{1 + e^{-x}}$$

Figure 8. Sigmoid function which serves as the activation function.

The output layer is the third layer of the neural network. It is trained using a process called mini-batch gradient descent optimization. Mini-batch gradient descent optimization is a variation of the gradient descent algorithm that divides the training dataset into smaller batches that are used to calculate the error and model coefficients. This method was chosen because it has a high update frequency and it allows for quicker convergence by reducing local minima. The batching also increases efficiency since not all training data is stored in memory and algorithm implementations. The downside was that error information must be accumulated across mini-batches of training examples but the decision was made that the high update frequency for this method was a worthwhile design concession. Ultimately, the mini-batch gradient descent optimization would continually make predictions on the training data and with each iteration, the neural network would optimize those predictions; significantly reducing error in the outputs. The error from mini-batch gradient descent and lack of multiple local minima can be observed below in Figure 9.

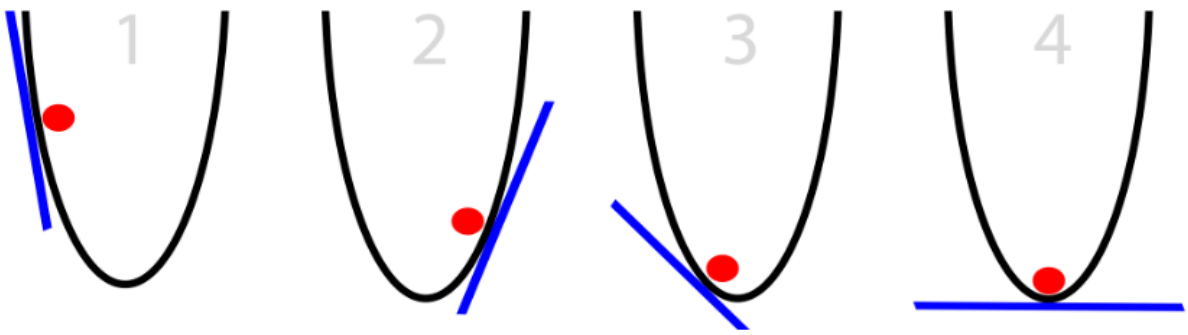

Figure 9. Plots of error for mini-batch gradient descent optimization.

The slope of the blue lines models the error so the Figure 9 plot 4 has the lowest error due to a slope of 0. The last part of the neural network error analysis comes from the optimizer which uses a cross-entropy loss function seen in Figure 10.

$$CE(\hat{y}, y) = -\sum_{i=1}^{n} y_i log(\hat{y}) + (1 - y_i) log(1 - \hat{y})$$

Figure 10. Cross-entropy loss function.

The cross-entropy loss function, also known as log loss, will measure the performance of the model since the output is a probability value between 0 and 1. The closer the probability is to 1 the lower the loss will be. This essentially measures the error in the slope for the optimizer. It is these input, output, and hidden layers that allow the neural network to replace the demodulator and channel

decoder. Together, the simulated conventional communication system and neural network architectures allow for the successful testing and demonstration of the artificial intelligence enabled wireless communication system.

Test Results

Testing and analyzing overall performance of the system can be quantified by three experiments which include plotting the bit error rate versus the number of training epochs, plotting normalized validation error against number of training epochs, and by plotting normalized validation error for code word lengths of 16, 32, and 64 bits.

Experiment 1 and 2 involve plotting bit error rate (BER) vs the number of training epochs and plotting the normalized validation error (NVE) vs the number of training epochs. The number of training epochs is the number of iterations the neural network will go through to train itself. Theoretically, as the number of training epochs increases, the BER and NVE should decrease dramatically. These two experiments were done for the Low Density Parity Check Codes (LDPC), Hamming (7, 4) Codes with zero padding, and Random Codes. To ensure that the results are robust for a range of situations the tests were repeated for four different signal-to-noise ratios (SNR) which include 20dB, 40dB, -60dB, -104dB, 15dB, 10dB, and 5dB.

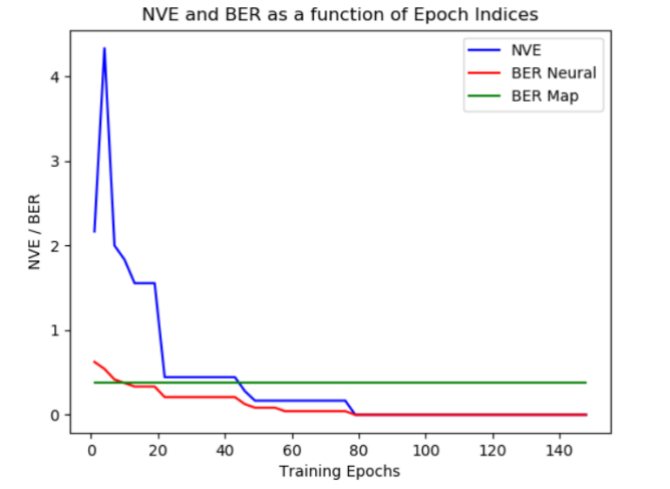

Figure 11. Plot of LDPC Code BER and NVE for various training epochs at a SNR of 20dB.

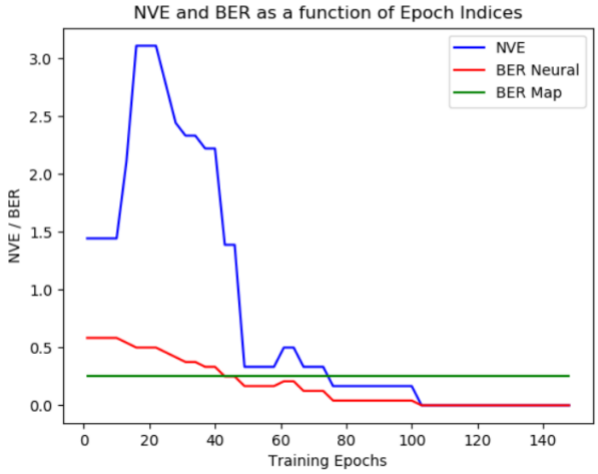

Figure 12. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of 20dB.

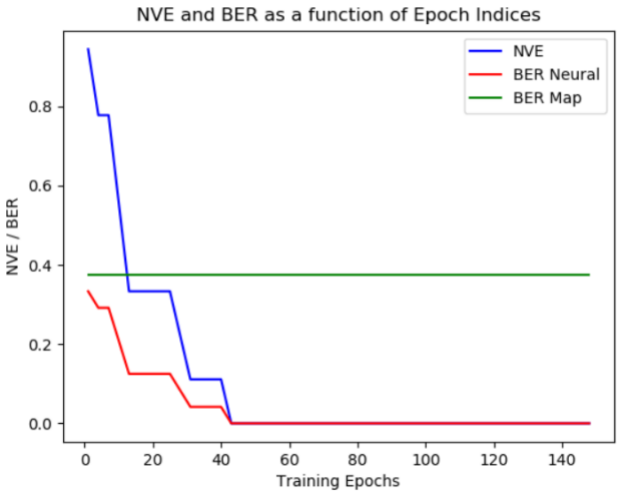

Figure 13. Plot of Random Code BER and NVE for various training epochs at a SNR of 20dB.

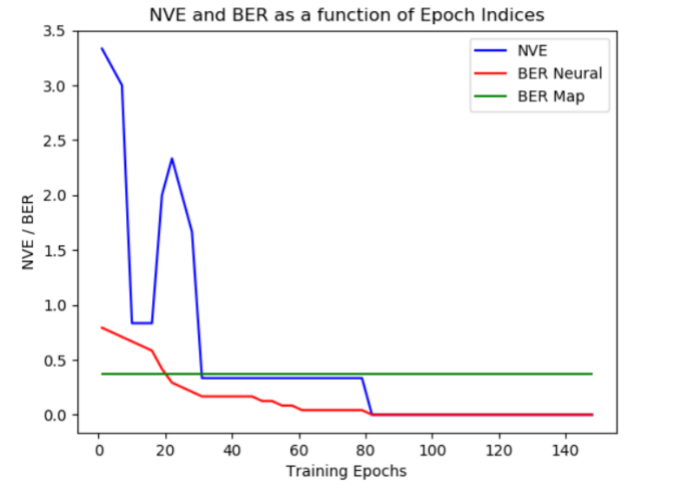

Figure 14. Plot of LDPC Code BER and NVE for various training epochs at a SNR of 15dB.

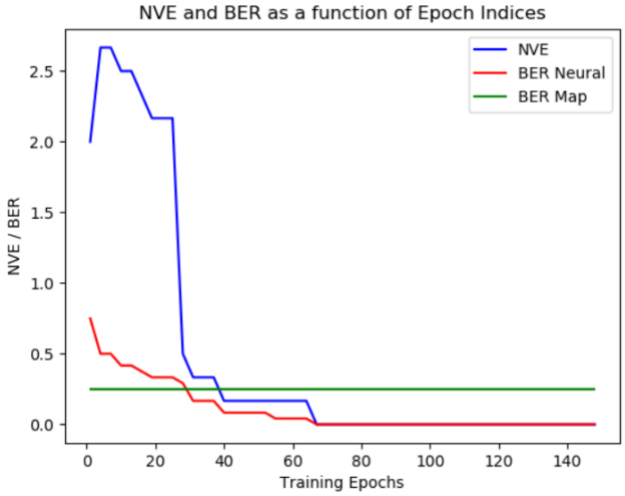

Figure 15. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of 15dB.

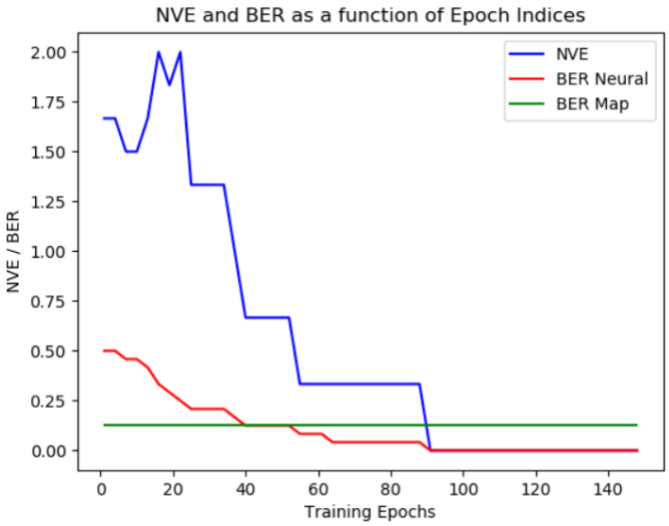

Figure 16. Plot of Random Code BER and NVE for various training epochs at a SNR of 15dB.

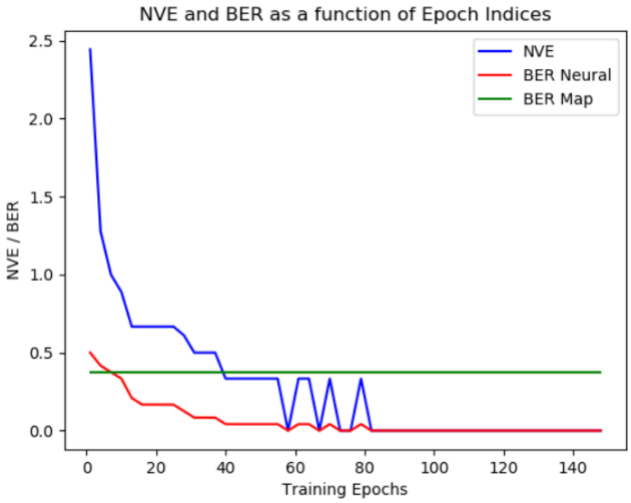

Figure 17. Plot of LDPC Code BER and NVE for various training epochs at a SNR of 10dB.

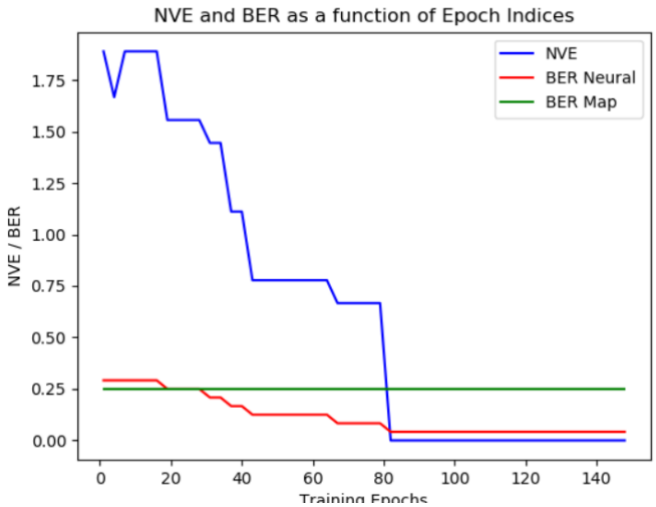

Figure 18. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of 10dB.

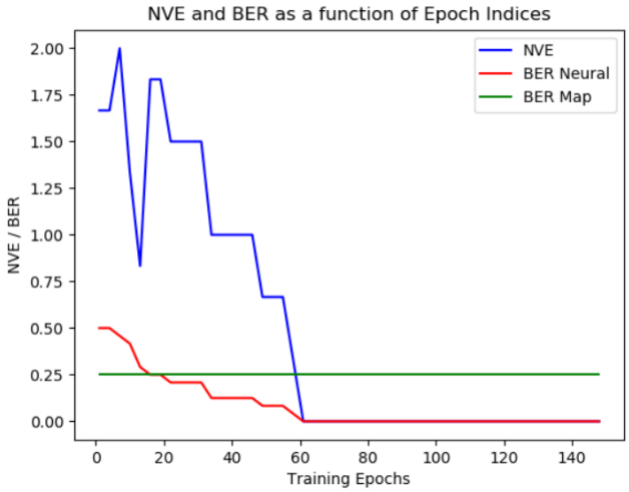

Figure 19. Plot of Random Code BER and NVE for various training epochs at a SNR of 10dB.

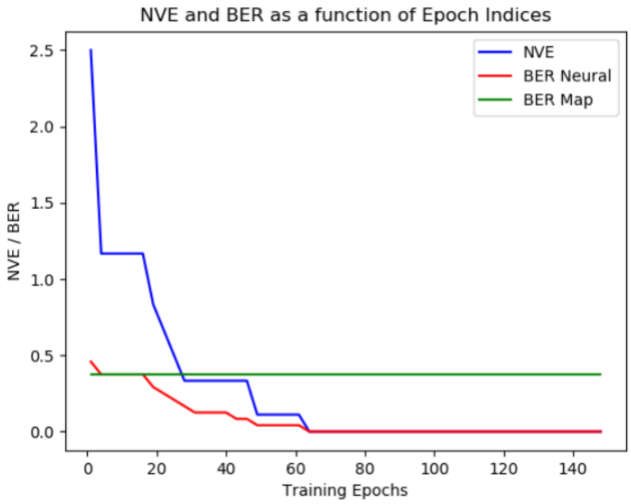

Figure 20. Plot of LDPC Code BER and NVE for various training epochs at a SNR of 5dB.

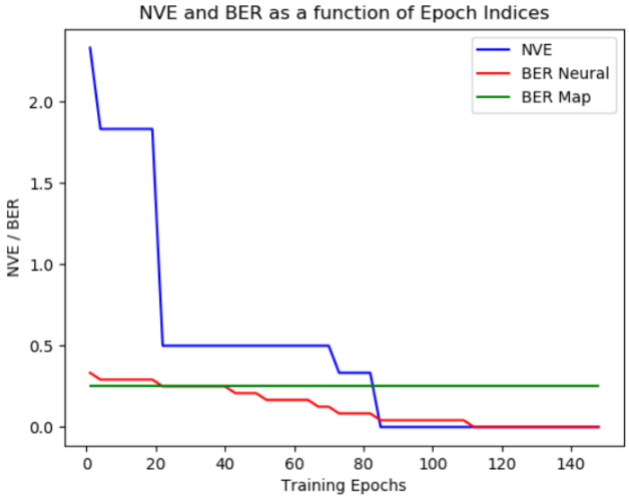

Figure 21. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of 5dB.

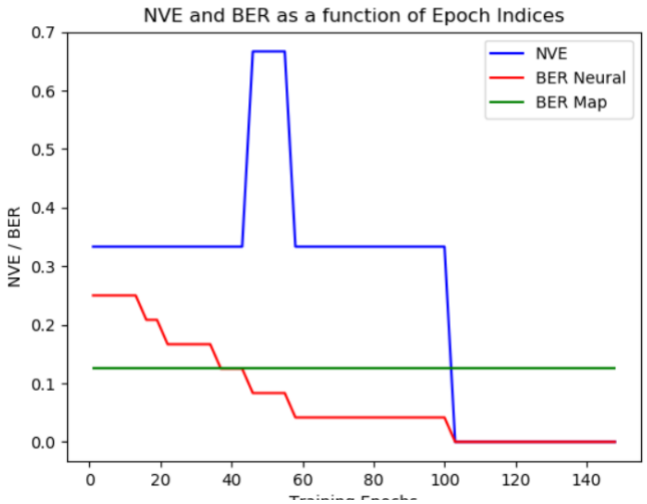

Figure 22. Plot of Random Code BER and NVE for various training epochs at a SNR of 5dB.

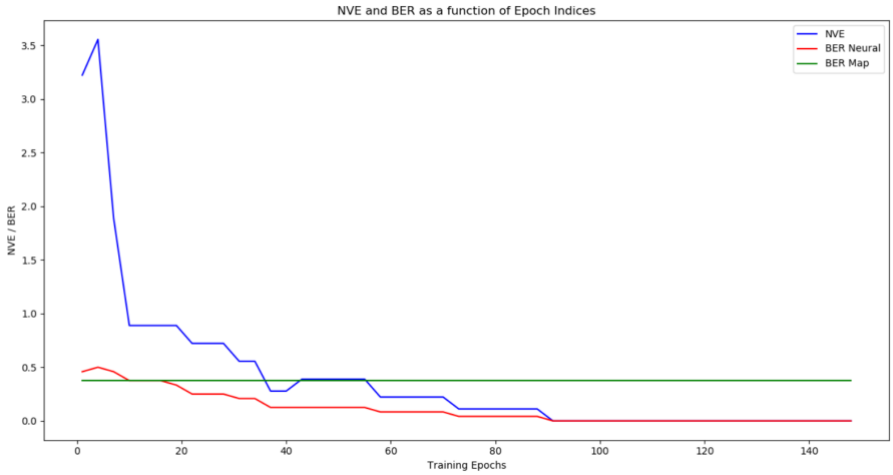

Figure 23. Plot of LDPC Code BER and NVE for various training epochs at a SNR of 40dB.

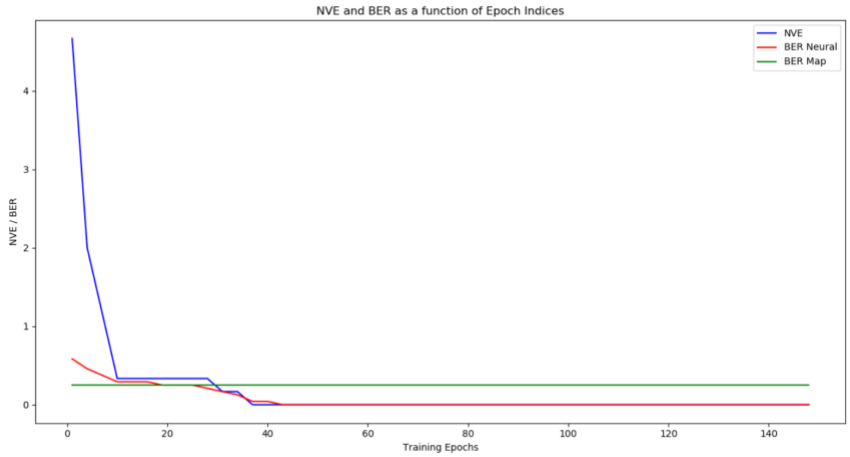

Figure 24. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of 40dB.

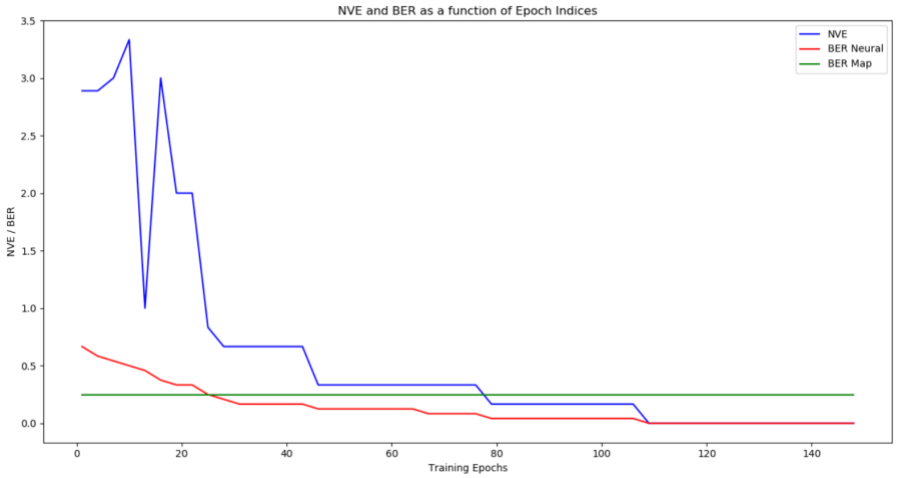

Figure 25. Plot of Random Code BER and NVE for various training epochs at a SNR of 40dB.

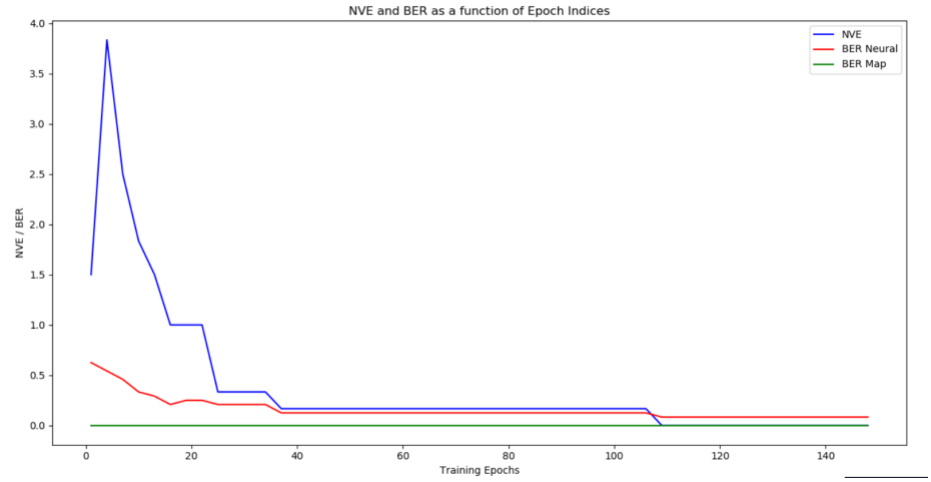

Figure 26. Plot of LDPC Code BER and NVE for various training epochs at a SNR of -60dB.

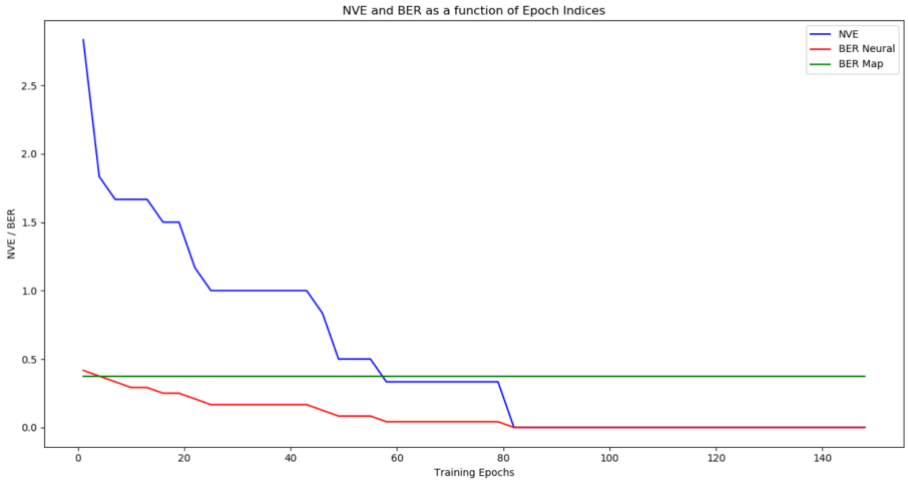

Figure 27. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of -60dB.

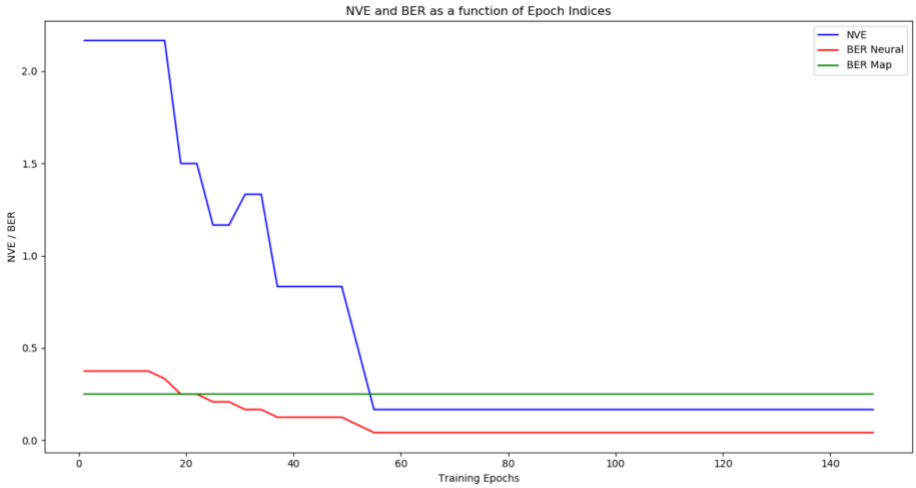

Figure 28. Plot of Random Code BER and NVE for various training epochs at a SNR of -60dB.

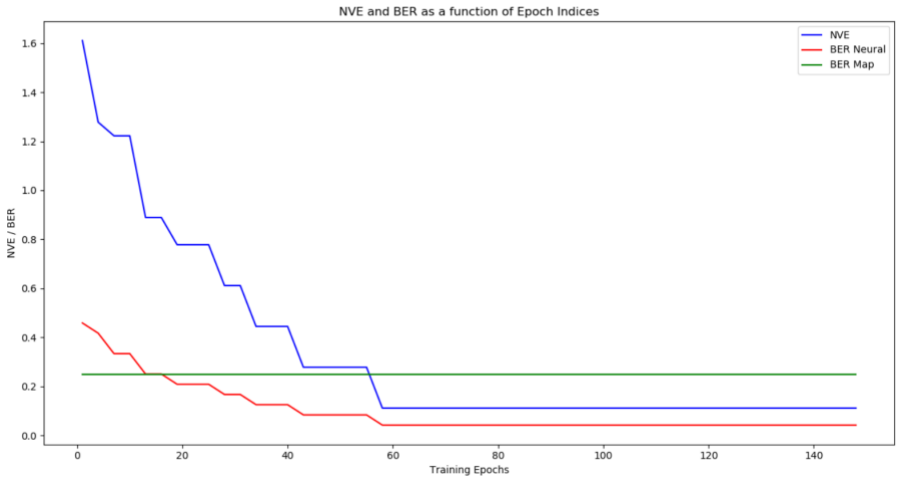

Figure 29. Plot of LDPC Code BER and NVE for various training epochs at a SNR of -104dB.

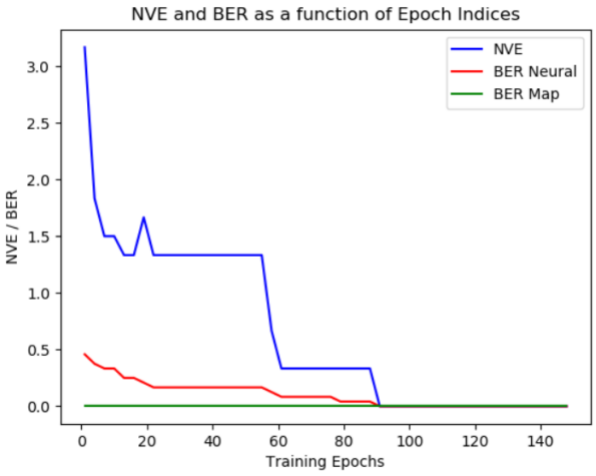

Figure 30. Plot of Hamming (7, 4) Code BER and NVE for various training epochs at a SNR of -104dB.

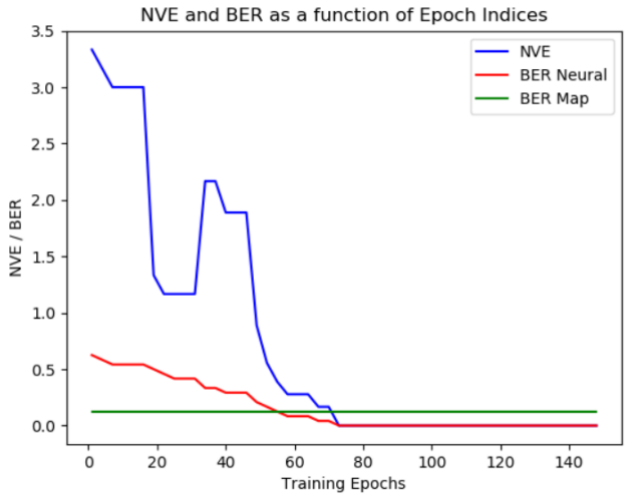

Figure 31. Plot of Random Code BER and NVE for various training epochs at a SNR of -104dB.

Note that a negative SNR value does not necessarily make sense; but the code is able to interpret a negative SNR argument at the expense of the original signal being dominated by noise. The data collected from tests run with a negative SNR values were included to highlight that the network can be over trained, and when the noise dominates the signal; the convergence of training for the neural network occurs at a slower rate. Also, at higher SNR levels, LDPC, followed by Hamming encoding perform the best. However, at medium to low SNR levels, the LDPC encoding scheme performs much better than Hamming. This is likely because by utilizing more bits than Hamming, LDPC codes can correct more errors at lower SNR levels, whereas Hamming can only correct 1-bit error at a maximum.

Looking at Figure 11 to Figure 22, it can be seen consistently that as the number of training epochs increases, the NVE and BER rates continue to decrease dramatically. The experiments yielded the best results at 20dB and 40dB which is expected since those SNR's mean that the signal is much greater than the amount of introduced noise. The plots for all three encoding schemes utilizing an SNR of -104dB resulted in the worst BER and NER because the amount of noise overwhelmed the signal strength. By visually inspecting the plots for experiments 1 and 2 it can be concluded that the system meets the performance metrics set by these experiments.

Experiment 3 utilized the LDPC Code encoding scheme to plot NVE and BER versus number of training epochs for code words of length 16, 32, and 64. This was done at a chosen SNR of 20dB.

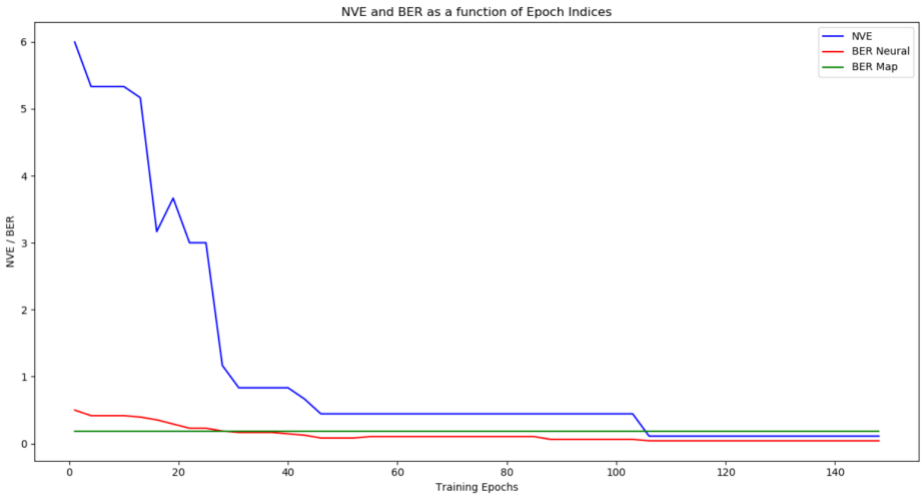

Figure 32. Plot of LDPC Code NVE and BER for code word of length 16.

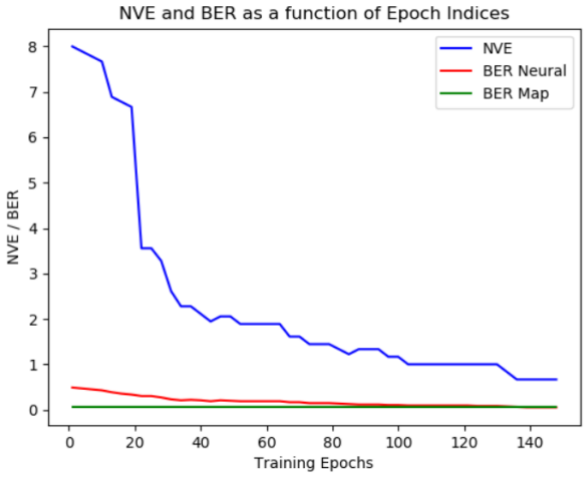

Figure 33. Plot of LDPC Code NVE and BER for code word of length 32.

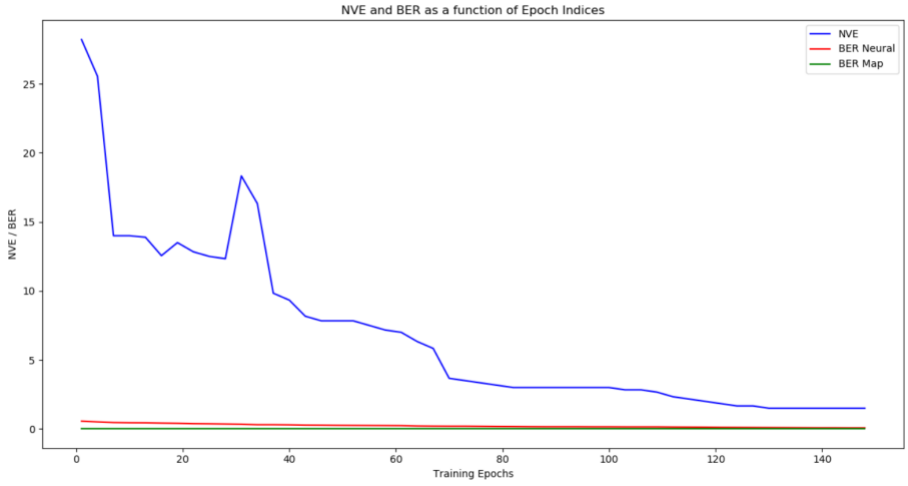

Figure 34. Plot of LDPC Code NVE and BER for code word of length 64.

Looking at these plots, it can be observed that for any given number of training epochs, the NVE and BER values are lower at lower code word lengths. This is expected since the longer a code word length, the greater of a chance for higher BER and NVE. Thus, the lowest errors should be seen at lower code word lengths.

Combining the theory from experiments 1, 2, and 3 leads to the conclusion that the lowest system performance error would occur at high SNR values and small code word lengths. Conversely, the highest system performance error would be expected at low SNR and high code word lengths. For the case of LDPC, the smallest code word of 16 and SNR of 104dB at a training epoch value of 30 yielded an NVE of about 1. For the LDPC case with the largest code word of 64 and SNR of 20dB at a training value of 30 yielded an NVE of about 13. Long code words had higher error because of a phenomenon called over training. This happened because the neural network trained on smaller length words and did worse on longer code words due to a lack of training exposure to words of that specification. Because all of our experimental plots follow the expected theoretical behavior it can be concluded that the neural network yields satisfactory levels of accuracy when demodulating and decoding the transmission sent to it.

Project Conclusion

Communication systems today have become very complex and are all around the world we live in. Traditional communication systems rely on static components that are hard coded and designed for specific circumstances. For example, a typical cell phone modem designed to work with one cellular provider using either GSM or CDMA technology. In this project the objective was to substitute the decoder and demodulator on a communication system with a neural network. Standard decoders and demodulators are designed for specific encoding schemes and fail when given a scheme it has not encountered before. The neural network is able to use training data and learn by comparing the inputted signal with known encoding schemes that best match. The best match criteria was determined by looking at which of the encoding schemes the neural network could choose from yielded the lowest probability of error. The neural network would then utilize that scheme as the best approximation for the original transmitted encoding scheme. When testing across three experiments, it was determined that the neural network does perform satisfactorily and meets the goals of the project. In future iterations, this neural network could be expanded to handle more types of encoding schemes and the probability of error could be further reduced. The ability to utilize artificial intelligence in communication systems represents a great advance in the industry and can further fuel innovative products that will enhance our daily lives.

Appendix

Code Files:

Note: tensorflow-1.6.0-cp36-cp36m-win_amd64.whl may need to be installed depending on user's installed python version.

1. main.py
    - Interface file to control the system.
2. requirements.txt
    - Stores libraries required to run all applicable code files.
    - The command "python -m pip install -r requirements.txt" will install the necessary libraries.
3. app
    - This folder contains the all necessary folders that make up the system.
4. Neural_Network
    - This folder has the implementation of the neural network in tensor.
5. data
    - This folder has the text files of ASCE UFTA encoded data used to send messages through the communication system.
6. encoder
    - This folder includes all the code that implements the encoder, modulation, demodulation, and decoder modules in a standard communication system.
    - In the modulation code, all modulation schemes are implemented here.
7. testcases
    - This folder includes sample test cases used to verify and test the conventional communication system and to train the neural network.
8. utility
    - This folder contains the performance functions to implement bit error rate (BER) and normalized validation error (NVE).

List of Figures:

How to Run the Program:

First, make sure to download the program from the repository.

In the directory /NeuralDemod/ install the necessary pre-requirements for your python installation using: python -m pip install -r requirements.txt

To run the program, you must be within the /NeuralDemod/app directory and run:

python -m run_app [Parameters]

To start a new training session on the neural-network:

Python -m run_app -t

To overwrite the existing waveform_samples.txt file after modifying the raw strings used in the textfiles in the /NeuralDemod/data folder, use

Python -m run_app -s

The SNR can also be specified with the parameter: -u [Integer]

The modulator can also be specified with the parameter: -m [QPSK_Modulator, BPSK_Modulator, QAM_Modulator]

The encoder used to decode and encode can be specified with: -e [LDPC, Hamming, Random]

To load the last training session (note that you should keep the previous parameters the same if using the same training session, and that you must also specify that you're training using the -t option): python -m run_app -t -v