Machine Learning in Radio Frequency Communications

Josh Booth, Trey Morris

***Abstract*— In this paper, we look at various metrics evaluating the ability of machine learning to classify codewords polluted with noise. Among these metrics are how well a model classifies codewords based on training time, how well a model classifies codewords at various E_b/N_o levels, and how codeword length affects classification time.**

I. Introduction

In a communication driven age, efficiency and reliability in radio frequency (RF) communications are of utmost importance. However, noise on a channel hinders these metrics through message corruption rendering the messages unreadable, causing a retransmission to be required. Recent work using a very popular class of error correction codes, Low-Density Parity-Check (LDPC) codes, has yielded promising results to drastically improve the efficiency and reliability of RF communications in noise. The LDPC codes used in this paper are deterministic π-rotation low-density parity check codes, which reduce the randomness associated with conventional LDPC codes using a single permutation vector to create the key defining the π-rotation pattern [1] (later referred to as m, a, b values.) These types of linear codes have been shown to allow a channel's capacity to approach the Shannon Limit, which is the theoretical maximum information transfer rate for a channel with a particular E_b/N_o level (noise to signal ratio.) However, LDPC codes rely on the "message passing algorithm" to decode messages, which is problematically a very formal process not allowing the algorithm to adapt to variable noise, causing poor results in dynamic environments. Thankfully, the message passing algorithm shows interesting similarity to previous studies in intelligent systems [2], such as machine learning, which is a "subset of artificial intelligence that uses statistical techniques to give computers the ability to 'learn' with data, without being explicitly programmed [3]." Essentially, it is a program that learns from its mistakes and can adapt to fix them, giving machine learning the ability to adapt to the variable noise in messages, unlike the message passing algorithm. The similarity between the two creates a bridge for implementing machine learning into the message passing algorithm to counter its formality. However, integrating machine learning into the algorithm blindly does not follow proper scientific procedures. Before implementing machine learning into a well-studied area of RF communications, it is worthwhile investigating and quantifying machine learning's ability to classify codewords (representations of a longer message compressed into a minimal number of bits) polluted with noise. This paper focuses on gathering and analyzing such data, allowing for future application of machine learning in areas of RF communications, such as integrating it into the message passing algorithm.

II. Creating the Network

A. *Creating Truth Data*

In order to create a machine learning network (also called a model) that accurately identifies codewords, the computer must first learn from "truth" data which are pre-labeled codewords the computer "learns" from by identifying patterns and sequences within the data. Traditionally, truth data is gathered from a pre-existing repository. However, since this is a newly explored application for machine learning, no repositories were aptly available, so we had to synthesize our own.

To ensure similarity between the data used in the message passing algorithm and machine learning process, we only used certain, well-studied m, a, b values [1] to create the codewords. The m, a, b values we used varied from information length 4 through 11. For clarification, information length refers to n in 2^n when determining the number of information bits in a codeword. We used a code rate of 50%, meaning 50% of the message is parity. A codeword of information length 4 is 32 bits long, with 2^4 information bits, and 2^4 parity bits. By generating codewords at different lengths, this allowed us to observe how codeword length affected the accuracy of classification.

To create our truth dataset, we first generated all possible codewords for a particular length using each length's respective m, a, b value and then labeled each codeword with its respective class number. For each codeword length, the domain of possible messages is 2^n messages when n is the number of information bits. The example in Table 1 helps to visualize what this would look like when $n = 1$.

TABLE 1
POSSIBLE CODEWORD CLASSIFICATIONS FOR 2^1 BITS

Class Number	Binary Codeword
1	XX**00**
2	XX**01**
3	XX**10**
4	XX**11**
0	Not Codeword
X = parity bit	

With 9 different lengths of codewords generated (length 4 through 11) and labeled with their respective class, we needed to add fake noise to the codewords so the machine learning algorithm could start to "learn" what a certain codeword might look like when polluted with noise. For this, we used additive white Gaussian noise (AWGN.) As seen in Figure 1, AWGN can be visualized as fairly constant, evenly distributed noise, which is steadier than real-world noise (dynamic noise.) The consistent nature of AWGN makes it very good for creating baseline data.

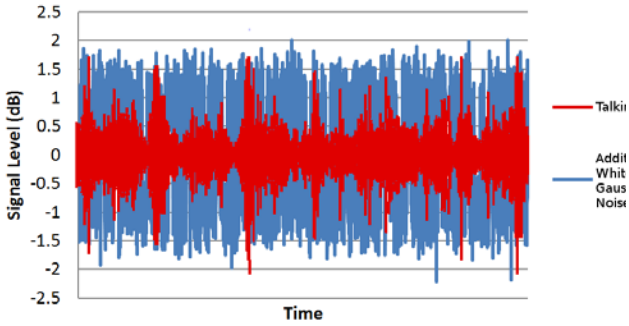

FIGURE 1. ADDITIVE WHITE GAUSSIAN NOISE VS. ENVIRONMENTAL NOISE

To add noise to a codeword, for each binary digit in the codeword a value from a Gaussian distribution of numbers was randomly selected and added to the digit equivalent of each binary number (1 or 0.) From there, we "sampled" the codeword to simulate transmitting it with a ±1 volt signal with -1 equaling 0 and +1 equaling 255, which placed all the noisy values between 0 and 255. We then repeated this process 1,000 different times per codeword to create a large sample of what a polluted codeword might look like. Lastly, we needed a sample of random Gaussian noise without a codeword so the machine learning algorithm would not learn to just look for a Gaussian distribution among the values. For each codeword length, we took the mean and standard deviation of all noisy samples to create a random Gaussian distribution of values. From there, we randomly sampled the Gaussian distribution to create random noise that still had the same general noise distribution and length as the polluted codewords, but did not contain an actual codeword and labeled it as a non-codeword. We then repeated this process 1,000 times per length to give each length 1,000 examples of non-codewords. We now had $\left((2^n + 1) * 1000\right)$ samples of truth data for each codeword length from n = 4 to n = 11 and were ready to create the machine learning architecture to train on our data.

B. *Machine learning Architecture*

A machine learning network can be thought of as part of a brain. There are neurons or, in machine learning nodes, that pass information between each other in a certain fashion dictated by the topology of the network. As messages are passed around, the synapses (weights, in machine learning) change, affecting the flow of information, determining how a network thinks and behaves. With machine learning, unlike a brain, we can create the network with specific types of nodes and topology so the network learns in a particular way.

The first architecture choice we made was the topology, as it determines the layout of nodes as well as how data flows through them. Throughout the years, many different types of architectures have arisen, each specializing in solving different types of problems. For our particular problem, where a string of data (or text) needs to be classified, a bidirectional recurrent neural network (BRNN) works best. BRNN's "use a finite sequence to predict or label each element of the sequence based on the element's past and future contexts [4]." Inside the BRNN, LSTM (Long Short-Term Memory) cells are traditionally used, which allows the model to remember past decisions when classifying data (in the brain analogy, these are the nodes.) However, we opted for the simpler, more computationally efficient, Gated Recurrent Unit (GRU) cells. They have only recently been developed, but have shown better results than LSTM cells for certain types of datasets [5]. This type of architecture, a BRNN with GRU cells, allows the network to classify a codeword while remembering nearby codewords, allowing for the model to use what can be thought of as context clues when classifying each codeword. For example, when attempting to classify a codeword, if the previous codeword had a relatively small amount of noise and the following codeword had a very large amount, the model can assume the codeword in question has a noise level somewhere in between the two values, allowing the model to account for the noise, which aids in dynamic classification.

After determining the architecture to use, we constructed it in the programming language Python using the TFLearn library, a wrapper built on top of Google's open-source machine learning framework, TensorFlow.

C. *Training the Models*

After the architecture was constructed, it was time to train the networks on the truth data. For each codeword length, a separate model was created to see how codeword length affects classification. Before creating the models, however, 1/3 of the truth data was set aside to be used as validation data after the model was trained. Validation data is used to check the model's training accuracy by hiding the label on the codeword, having the trained model predict the

codeword, then comparing the label to the model's prediction.

Initially, the truth data was input into the network 200 times (epochs) to assure the network had plenty of time to iterate through and learn from the data. This is where the network's weights (synapses) are automatically updated as the model learns. While we initialized the weights of the GRU cells with the length of the codeword, the model adjusts the weights as it learns.

The results of testing the model with the validation data, as seen in Table 2, shows that as the number of bits in a codeword increases, so does the accuracy. This offers promise for future application, due to the fact that using codewords in this manner only becomes practical once *n* equals at least 12.

TABLE 2
ACCURACY OF VALIDATION DATA USING 200 EPOCHS

Information Length	Validation Accuracy (200 epochs)
4	87%
5	90%
6	96%
7	96%
8	97%
9	98%
10	99%
11	99%

The results were very good for most codeword lengths; however, this was after a long training session. The quicker the models train, the more robust they can be, as they can be quickly retrained if need be. We retrained the data using only 75 epochs, cutting the training time to less than half the original while retaining very strong results at higher information bit lengths (Table 3.)

TABLE 3
ACCURACY OF VALIDATION DATA USING 75 EPOCHS

Number Of Information Bits in Codeword	Validation Accuracy (75 epochs)
4	73%
5	57%
6	88%
7	92%
8	92%
9	98%
10	98%
11	99%

III. TESTING THE NETWORK

After creating the models, the next step was to test them when classifying codewords in different levels of E_b/N_o (noise,) allowing us to observe not only if the models are able to classify codewords at noisier levels, but also see if they have truly generalized how to identify noisy codewords. To test how the models handle varying E_b/N_o levels, we took each codeword length in our truth data and added additional noise to the truth data every half step from 5 E_b/N_o to -2 E_b/N_o, creating a total of 15 variations of each length codeword. Lastly, the labels were removed from all the data so the models would have to classify the data solely based on what they have learned (much like validation data.) After the models predicted the classifications of the codewords, the predictions were then formatted into a confusion matrix. In machine learning, a confusion matrix is a table used to identify the accuracy of a particular model. Along the X-axis is the model's prediction, and along the Y-axis is the truth data. Table 4 is an excerpt of the first 6 classes of a confusion matrix from information length 8 when E_b/N_o equaled 4. When a model predicted a codeword it thought was of class 2, and the model was right, the location (2, 2) was incremented. However, if the model thought the codeword was of class 2, but the codeword was actually of class 1, the location (2, 1) was incremented.

TABLE 4
CONFUSION MATRIX EXCERPT FOR LENGTH 8 (EPOCH 75)

E_b/N_o: 4	Right: 90.9% (233486)			Wrong: 9.1% (23514)			
	0	**1**	**2**	**3**	**4**	**5**	**6**
0	125	0	0	0	0	0	0
1	1	981	6	15	0	4	0
2	0	4	918	6	14	3	6
3	0	3	12	960	13	8	1
4	0	0	0	2	942	0	1
5	6	1	0	8	0	970	9
6	9	0	1	0	1	1	945

For demonstrative purposes, a codeword with medium accuracy was used to help better understand what happens when a codeword is incorrectly classified. Table 5 is an excerpt of a confusion matrix of codeword length 11 at E_b/N_o -2, which is a very low and thus noisy E_b/N_o. The drastically improved results

at a higher information length shows promise in implementing machine learning into radio frequency communications, as codewords polluted with lower E_b/N_o levels should be harder to classify in. However, in both charts, class 0 (non-codeword) is of lower accuracy than other rows, meaning the models' ability to recognize a codeword is noticeably worse than its ability to identify the class of a codeword.

TABLE 5
CONFUSION MATRIX EXCERPT FOR LENGTH 11 (EPOCH 75)

E_b/N_o: -2	Right: 99.3% (2035406)				Wrong: 0.7% (13594)		
	0	**1**	**2**	**3**	**4**	**5**	**6**
0	120	0	0	1	1	0	2
1	0	998	0	0	1	0	0
2	1	0	999	0	0	0	0
3	1	0	0	995	0	0	0
4	1	0	0	0	994	0	0
5	1	0	0	0	0	992	0
6	2	0	0	0	0	0	991

The percentages in the confusion matrices across the board were promising, so we opted to focus on the 75 epoch models, as they are more robust than the 200 epoch ones. Now that we had proof machine learning could classify codewords in noise, we wanted to quantify exactly how much poorer the 75 epoch models did as E_b/N_o decreased. We graphed the accuracy across all E_b/N_o values used (see Figure 2) to check if the models still predicted well in more intense noise, or if they only predicted well in a higher E_b/N_o.

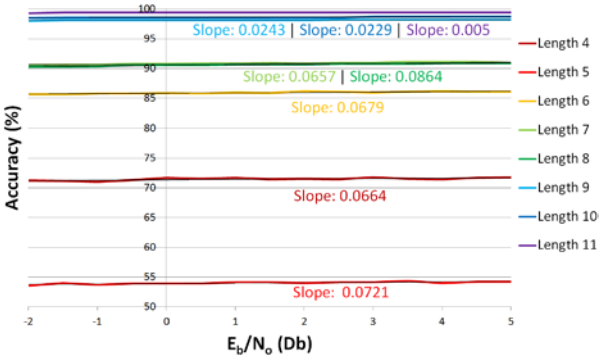

FIGURE 2. MODEL ACCURACY FOR VARYING E_B/N_O

The graph shows a slight downward slope as the E_b/N_o decreases, although the slope is nearly negligible, showing how little the amount of noise affects the model's ability to classify codewords. We now had results showing the models could predict well in almost any E_b/N_o level, including at E_b/N_o levels close to the Shannon Limit (-2 E_b/N_o is starting to approach the limit.)

It is obvious accuracy increases with codeword length; however, quantifying this increase in accuracy as the codeword length increases gives insight as to possible limits in the increase of accuracy.

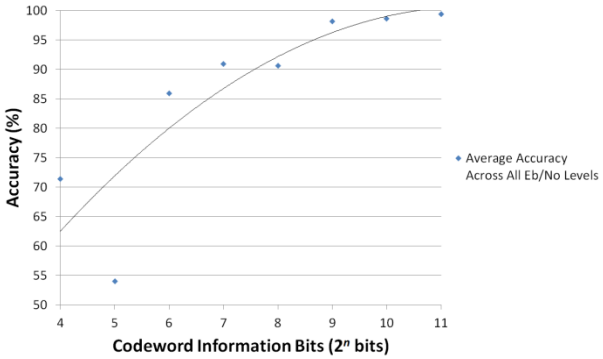

FIGURE 3. E_B/N_O ACCURACY BASED ON CODEWORD LENGTH

As shown in Figure 3, the increase in accuracy as codeword length increases is on a polynomial scale. To find the equation at which accuracy per codeword length increases, we took the average accuracies across the predictions for all E_b/N_o values, giving the equation $y = -0.6651x^2 + 15.394x + 11.584$. The rate of the increase in accuracy slowly diminishes as codeword length increases, showing that at some point, an increase in codeword length will no longer offer an increase in accuracy. However, with the upper codeword lengths tested here, specifically 9, 10 and 11, 100% accuracy is already being approached, which may nullify the need for much longer codewords.

IV. CONCLUSIONS

Proven in this paper was machine learning's ability to classify codewords after being polluted with noise at various E_b/N_o levels, allowing for future application in RF communications.

Eventually, the models can be transferred to Field Programmable Gate Arrays (FPGA) for use in live codeword detection. While classification speed would become the largest concern in live codeword classification, Figure 4 shows the speed of decoding linearly increases with codeword size, which should be taken care of by the speed of the FPGA.

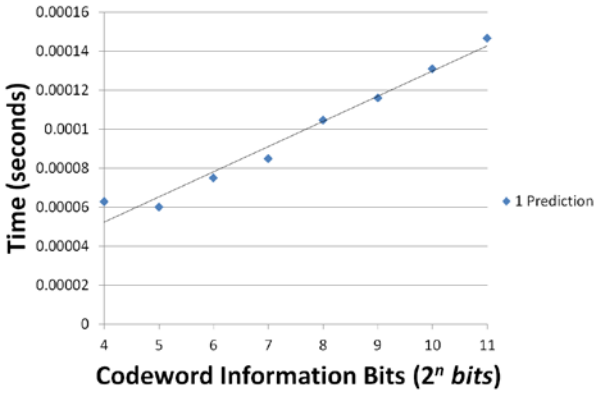

FIGURE 4. TIME PER CODEWORD PREDICTION

The future application of machine learning in RF communications offers enormous potential, from classifying codewords transmitted through noise on live FPGA's to implementing it into already established transmission techniques, such as the message passing algorithm. With so many applications of machine learning into RF communications, the future of communications with machine learning is promising.

ACKNOWLEDGMENTS

Josh Booth would like to acknowledge support from the Naval Research Laboratory through the STEM Student Employment Program.

REFERENCES

[1] R. Echard and S. C. Chang, "The pi-rotation low-density parity check codes," Proceedings of GLOBECOM 2001, pp. 980-984, November 2001

[2] Judea Pearl, "Probabilistic reasoning in intelligent systems: Networks of Plausible Interference," Morgan Kaufman Publishers, Inc., San Francisco, 1988.

[3] Samuel, A. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3), pp.210-229.

[4] Schuster, M. Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), pp.2673-2681.

[5] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2018*). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. [online] Arxiv.org. Available at: https://arxiv.org/abs/14123555 [Accessed 15 Aug. 2018]