Information and Coding Theory
March 28, 2021

Cooperative Learning of Encoding/Decoding Functions

Kodirjon Akhmedov , Meixing Liao

1 Abstract

In this project, several encoding/decoding approaches were studied. The experiments on different algorithms were implemented by various tools. Furthermore, the encoder and decoder were presented as NNs and joint training was performed. The performance was simulated by the given parameters and the corresponding plots were generated. NN implementation compared with repetitions codes using soft/hard decision decoding and on varying epochs NN trained and performance curves were illustrated. Our codes can be found in Github.

2 Introduction

Forwarding error correction or channel coding is a technique used for controlling errors in data transmission over unreliable or noisy communication channels has been interesting to many researchers and approaching towards the encoding and decoding using neural networks has been on the modern interest due to neural networks can learn a form of decoding algorithm, rather than only a simple classifier as well as neural networks have been used because of their adaptive learning, self-organization, and real-time operation[5].

This project implements the NN approach of encoding and decoding of channel coding and discusses its results also reviews past related works and their importance in the current discussion. Moreover, this project covers non neural network implementation of error correction codes as well as neural network implementation where random data generate and encoding, BI-AWGN channel, BPSK modulation and decoding processes were investigated. Furthermore NN results compared with repetition codes using both soft and hard decision decoding, beside that different epochs NN performance curves were also illustrated and conclusions were drawn [1].

3 Related Work

During the project implementation, we have reviewed the most significant research articles and used references provided by teaching assistants. When implementing NN, most important paper was On Deep Learning-Based Channel Decoding[1] where we studied good methods implemented by authors for comparison of polar and random codes using neural networks.We have read other more papers and thesis of students based on error correction codes[2], BER Analysis of BPSK for Block Codes and Convolution Codes Over AWGN Channel[3], Performance comparison of short-length error-correcting codes, and Decoding of Error Correcting codes Using Neural Networks[4]'. After careful investigation we have started our project's practical preliminary and NN implementations.

[0] Kodirjon.Akhmedov@skoltech.ru
[0]Meixing.Liao@skoltech.ru

4 Preliminaries and preparations

4.1 Preliminaries

4.1.1 BPSK Modulation

Binary Phase Shift Keying (BPSK) is a two phase modulation scheme, it's the simplest form of phase shift keying. It uses two phases which are separated by 180°. The pairs of the signals $S_1(t)$ and $S_2(t)$ are used to represent binary symbols 1 and 0 respectively.

For binary 0:

$$s_0(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi ft + \pi) = -\sqrt{\frac{2E_b}{T_b}}\cos(2\pi ft) \tag{1}$$

For binary 1:

$$s_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi ft) \tag{2}$$

where E_b means energy per bit, T_b represents the bit duration.

Figure 1 provides an example of BPSK modulation.

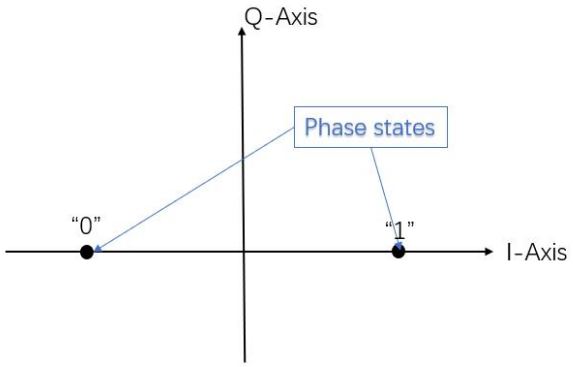

Figure 1: The example of BPSK Modulation

4.1.2 BI-AWGN Channel

Additive White Gaussian noise channel is the most common type of noise added over the channel.In practice, many types of noise sources are additive and independent of each other. Thus, by the Central Limit Theorem, when added the noises together, it can be approximated by a zero-mean Gaussian random variable with some variance denoted as σ^2.

Consider the case of BI-AWGN channel(figure 2, credit:Tomáš Filler), the random variable $X \in \{-1, +1\}$. And the output $Y = X + Z$, where Z is a zero-mean Gaussian random variable with variance σ^2.

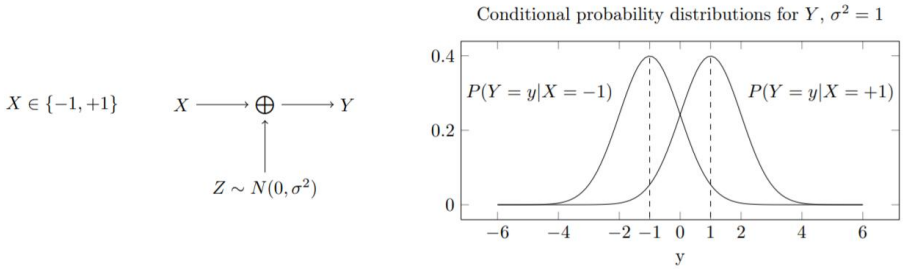

Figure 2: Binary Additive White Gaussian Noise Channel(BI-AWGN Channel)

Hence, we have the following conditional power density functions of Y:

$$P(Y = y \mid X = +1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-1)^2}{2\sigma^2}\right] \tag{3}$$

$$P(Y = y \mid X = -1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y+1)^2}{2\sigma^2}\right] \tag{4}$$

4.1.3 Different Error Correcting Codes

During our lectures, several error correction codes were discussed.

Hamming Code

Hamming code is a kind of linear error-correcting code. It can detect up to 2-bit errors or correct 1-bit errors without detection of uncorrected errors. The parameters of Hamming codes are as follow:

- Block length: $n = 2^m - 1$
- Message length: $k = 2^m - m - 1$
- Distance: $d = 3$
- Rate: $R = 1 - \frac{m}{2^m - 1}$

The steps of decoding with a binary Hamming code are as follow:

1. When $\mathbf{w}$ is received, calculate its syndrome $S(\mathbf{w}) = \mathbf{w}H^{\mathrm{T}}$.
2. If $S(\mathbf{w}) = \mathbf{0}$, assume $\mathbf{w}$ was the codeword sent.
3. If $S(\mathbf{w}) \neq \mathbf{0}$, then $S(\mathbf{w})$ is the binary representation of j, for some $1 \leq j \leq 2^r - 1$. Assuming a single error, the word $\mathbf{e}_j$ gives the error, so we take the sent word to be $\mathbf{w} - \mathbf{e}_j$ (or, equivalently, $\mathbf{w} + \mathbf{e}_j$.

Reed Muller Code

Consider the boolean function from m variables with degree no more than r. The coefficients corresponding to information bits. A value of this polynomial in all points of m -dimensional Boolean cube is a codeword (evaluation code). This code is called a $RM(m, r)$ code:

$$RM(m, r) = \{f : \{0, 1\}^m \to \{0, 1\} \mid \deg f \leq r\} \tag{5}$$

The Reed-Muller $\mathrm{RM}(r, m)$ code of order r and length $N = 2^m$ is the code generated by v_0 and the wedge products of up to r of the $v_i 1 \leq i \leq m$ (where by convention a wedge product of fewer than one vector is the identity for the operation). In other words, we can build a generator matrix for the $\mathrm{RM}(r_1 m)$ code, using vectors and their wedge product permutations up to r at a time $v_0, v_1, \ldots, v_n, \ldots, (v_{i_1} \wedge v_{i_2}), \ldots (v_{i_1} \wedge v_{i_2} \ldots \wedge v_{i_r})$, as the rows of the generator matrix, where $1 \leq i_k \leq m$. For $1 \leq i \leq m$, let $\mathbf{v}_i$ be a binary 2^m -tuple of the following form:

$$\mathbf{v}_i = (\underbrace{0 \cdots 0}_{2^{i-1}}, \underbrace{1 \cdots 1}_{2^{i-1}}, \underbrace{0 \cdots 0}_{2^{i-1}}, \cdots, \underbrace{1 \cdots 1}_{2^{i-1}})$$

which consists of 2^{m-i+1} alternating all-zero and all-one 2^{i-1} -tuples.

The encoding of Reed-Muller code can be also described by Kronecker product:

$$c = u \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes m} \tag{6}$$

where u is the information sequence, and the Kronecker product is defined as:

$$A \otimes B = \begin{bmatrix} a_{1,1}B & a_{1,2}B & \cdots \\ a_{2,1}B & a_{2,2}B & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{7}$$

$$A^{\otimes n} = A \otimes A^{\otimes(n-1)} = A^{\otimes(n-1)} \otimes A \tag{8}$$

The procedure of RM decoding is as follow:

1. We receive $y = x + e, x \in RM(m, r)$
2. For each monomial x^σ of degree r calculate $\phi_A(y)$ over 2^{m-r} assosiated facets $A(x^\sigma) = \{A_1, \ldots, A_{2^{m-r}}\}$
3. $a_\sigma \leftarrow \text{maj}\left\{\phi_{A_1}(y), \phi_{A_2}(y), \ldots, \phi_{A_{2^{m-r}}}(y)\right\}$
4. Remove monomials of degree r, i.e. $y' = y - \text{enc}\left(\sum a_\sigma \text{x}^\sigma\right) = x' + e, x' \in RM(m, r-1)$
5. Continue with $r \leftarrow r - 1, y \leftarrow y'$

Reed Solomon Code

Reed Solomon Code is a kind of MDS code.

First, it was suggested to construct polynomial code. Than the distance was calculated and people found that codes are MDS codes and it was very good.

The Reed-Solomon code $RS(n, k), 1 \leq k \leq n$, is defined as follow:

$$RS(n,k) = \{(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) : f(x) \in \mathbb{F}_q[x], \deg f(x) < k\} \tag{9}$$

It has length $n \leq q$, dimension k and $d = n - k + 1$.

To encode, we have $f = (f_0, f_1, \cdots, f_{n-1})$ as vector of coefficients of f(x).

$$c = f\mathbf{G} \tag{10}$$

where $\mathbf{G}$ is the generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ {\alpha_1}^{k-1} & {\alpha_2}^{k-1} & \ldots & {\alpha_n}^{k-1} \end{bmatrix} \tag{11}$$

We can also present by polynomials:

$$c(x) = f(x)g(x) \tag{12}$$

Where $g(x) = (x - \alpha)\left(x - \alpha^2\right) \cdots \left(x - \alpha^{d-1}\right)$ is the generator polynomial.

We can use Peterson-Gorenstein-Zierler decoding algorithm to decode. Let us consider the decoding of $[n, k, d = n - k + 1]$ RS code, let $\mathbf{v}$ be the received sequence.

1. Calculate syndrome $\mathbf{S} = (S_1, S_2, \ldots, S_{2t}) = \mathbf{H}_v^T$
2. Find the actual number of errors r as the largest L for which $\det M_L \neq 0$.
3. Find $\sigma_r, \sigma_{r-1}, \ldots, \sigma_1$ from the system

$$M_r \left[\sigma_r, \sigma_{r-1}, \ldots, \sigma_1\right]^T = -\left[S_{r+1}, S_{r+2}, \ldots, S_{2r}\right]^T \tag{13}$$

4. (Chien search) We know $\sigma(z)$, find X_i by exhaustive search over all the elements of $\mathbb{F}_q$. Find error positions from error locators.
5. Find Y_i from the system

$$\begin{bmatrix} X_1 & X_2 & \cdots & X_r \\ \vdots & \vdots & \ldots & \vdots \\ X_1^r & X_2^r & \cdots & X_r^r \end{bmatrix} \begin{bmatrix} Y_1 \\ \vdots \\ Y_r \end{bmatrix} = \begin{bmatrix} S_1 \\ \vdots \\ S_r \end{bmatrix} \tag{14}$$

Cyclic Code

We say a code $\mathcal{C}$ is cyclic if all cyclically shifted words are also valid code words of this code.

g(x) is a generator polynomial of C if g(x) is a normalized polynomial of smallest degree in C.Let $g(x)$ be the generator polynomial of a cyclic code C of length n. If $g(x)$ has degree $n - k$, then the codewords $g(x), xg(x), \ldots, x^{k-1}g(x)$ clearly form a basis for C, i.e. C is an $[n, k]-$ code.

Hence, if $g(x) = g_0 + g_1 x + \cdots + g_{n-k}x^{n-k}$, then

$$G = \begin{bmatrix} g_0 & g_1 & \dots & g_{n-k} & 0 & 0 & \dots & 0 \\ 0 & g_0 & \dots & g_{n-k-1} & g_{n-k} & 0 & \dots & 0 \\ 0 & 0 & \dots & & & & \dots & 0 \\ 0 & 0 & \dots & & g_0 & g_1 & \dots & g_{n-k} \end{bmatrix} \tag{15}$$

is a generator matrix for C.

For cyclic code, we have $g(x) \mid x^n - 1$.We call h(x) the check polynomial of C:

$$h(x) = \frac{x^n - 1}{g(x)} \tag{16}$$

Then we can obtain the parity check matrix:

$$H = H_{(n-k)\times n} = \begin{pmatrix} h_k & h_{k-1} & \dots & h_0 & 0 & 0 & \dots & 0 \\ 0 & h_k & \dots & h_1 & h_0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & h_k & h_{k-1} & \dots & h_0 \end{pmatrix} \tag{17}$$

For non-systematic encoding:

$$c(x) = a(x)g(x) \tag{18}$$

While for systematic:

$$\begin{gathered} a'(x) = x^{n-k}a(x) = g(x)b(x) + r(x) \\ c(x) = a'(x) - r(x) \in \mathcal{C} \end{gathered} \tag{19}$$

To decode, we need to calculate the syndrome:

$$S(x) = S_{y(x)} = y(x) \mod g(x) \tag{20}$$

BCH Code

A BCH code of designed distance d is a cyclic code over with generator polynomial equal to the least common multiple of the miminal polynomials of where is a primitive nth root of unity.

$$g(x) = LCM\left(m_b(x), \dots, m_{b+d-2}(x)\right)$$

Parameters:

-n: length

-d: designed distance

-m: minimal number such that $n \mid q^m - 1$

$$\exists \beta \in \mathbb{F}_q^* : |\beta| = n$$

The Parity check matrix:

$$H = \begin{pmatrix} 1 & \beta^b & \left(\beta^b\right)^2 & \dots & \left(\beta^b\right)^{n-1} \\ 1 & \beta^{b+1} & \left(\beta^{b+1}\right)^2 & \dots & \left(\beta^{b+1}\right)^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \beta^{b+d-2} & \left(\beta^{b+d-2}\right)^2 & \dots & \left(\beta^{b+d-2}\right)^{n-1} \end{pmatrix} \tag{21}$$

The decoding is using the same method as RS code.

Convolutional Code

Consider the general rate:$R = \frac{b}{c}$, we define the information sequence:

$$\mathbf{u} = \mathbf{u}_0\mathbf{u}_1 \ldots = u_0^{(1)}u_0^{(2)} \ldots u_0^{(b)}u_1^{(1)}u_1^{(2)} \ldots u_1^{(b)} \ldots \tag{22}$$

And the encoding sequence is split into blocks of length c:

$$\mathbf{v} = \mathbf{v}_0\mathbf{v}_1 \ldots = v_0^{(1)}v_0^{(2)} \ldots v_0^{(c)}v_1^{(1)}v_1^{(2)} \ldots v_1^{(c)} \ldots \tag{23}$$

where

$$\mathbf{v}_t = f\left(\mathbf{u}_t, \mathbf{u}_{t-1}, \ldots, \mathbf{u}_{t-m}\right) \tag{24}$$

The next $\mathbf{V_t}$ can be defined as some function of the previous m information blocks, here m is the encoder memory.

The encoding can be presented as:

$$\mathbf{v} = \mathbf{uG} \tag{25}$$

$$\boldsymbol{G} = \begin{pmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & \ldots & G_m & & \\ & G_0 & G_1 & \ldots & G_m & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{26}$$

The encoding process can be presented by code tree,finite state machine and trellis.

There are Bit-wise MAP and Block-wise MAP decoding. Block-wise decoder should return the most probable codeword:

$$\hat{x}^{(MAP)}(y) = \arg\max_{x \in \mathcal{C}} \Pr(x \mid y) \tag{27}$$

Bit-wise decoder just return the most probable value of the bit. We do it for all the bits. The result of bit wise decoding maybe be not belongs to the code.

$$\hat{x}_i^{(MAP)}(y) = \arg \max_{x_i \in \{+1,-1\}} \sum_{x \in \mathcal{C}, x_i \text{ is fixed}} \Pr(x \mid y) \tag{28}$$

Consider Viterbi algorithm:

Assume we send a codeword $\mathbf{x} \in \mathbb{F}_2^n$ and received a sequence $\mathbf{y}$. Recall, that the channel is memoryless, then

$$\Pr(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{n} \Pr\left(y_i \mid x_i\right) \tag{29}$$

Label the edges of trellis with $\Pr\left(y_i \mid x_i\right)$ values. To find the probable pass, we go to the last state, find all possible paths which go to this state, and then choose the path with the largest matrix.

LDPC Code

LDPC codes were designed by Robert G. Gallager in the 1960s and forgotten for three decades. LDPC codes functionally are defined by a sparse parity-check matrix. For LDPC codes, we have:

$$R \geq 1 - \frac{J}{K} \tag{30}$$

where J and K correspond to the column weight and row weight.

Erasure-correction algorithm, Bit-flipping algorithm, Message-passing algorithm and Belief propagation can be used to decode.

Polar Code

Polar codes are the first explicit capacity-achieving construction. It's invented by Arikan in 2009.Polar code is a linear code, thus to perform the encoding, one needs to construct a generator matrix. The information is transferred only through "good" channels. For "bad" channels, one just fixes the value of bits to transmit. Bit positions corresponding to "bad" channels are frozen.

The decoding method decodes a single bit at each step, known as a successive cancellation (SC) decoding:

1. Successive cancellation decoding makes a hard decision at each decoding step, thus, one can make two decisions and keep two hypothesis about the decoded bit value.
2. There are 2^k hypotheses at step $k \Rightarrow$ hypothesis count grows exponentially.
3. To reduce the overall complexity, one can keep $L \ll 2^k$ most probable hypotheses (or paths).
4. To make a choice among L paths, the CRC checks are added.

4.2 BER of BPSK in BI-AWGN Channel without coding

The bit error rate (BER) is the number of bit errors per unit time, it equals to the number of bit errors divided by the total number of transferred bits during a studied time interval.
In the BPSK-AWGN Channel model:

$$P(X=-1)=P(X=1)=\frac{1}{2} \tag{31}$$

Thus:

$$\text{BER}=P(X=-1)\times P(-1 \text{ transfer as } 1)+P(X=1)\times P(1 \text{ transfer as } -1) \tag{32}$$

As $P(X=1)=P(X=-1)$ and $P(-1 \text{ transfer as } 1)=P(1 \text{ transfer as } -1)$, the function (6) can be simplified as:

$$\begin{aligned}\text{BER} =& 2\times P(X=1)\times P(1 \text{ transfer as } -1) &(33)\\ =& 2\times\frac{1}{2}\int_t^{\infty}\frac{1}{\sqrt{2\pi}\sigma}\times\exp(-\frac{(t-\mu)^2}{2\sigma^2})dt &(34)\\ =& \int_t^{\infty}\frac{1}{\sqrt{2\pi}}\times\exp(-\frac{t^2}{2})dt &(35)\end{aligned}$$

where t is the power of the symbol.
By the definition of SNR:

$$\text{SNR}=\frac{\text{Eb}}{\text{N0}} \tag{36}$$

Also we have:

$$\sigma^2=\frac{\text{N0}}{2} \tag{37}$$

Thus:

$$\begin{aligned}\text{Eb} =& \text{SNR}\times\text{N0} &(38)\\ =& \text{SNR}\times 2\sigma^2 &(39)\\ =& 2*\text{SNR} &(40)\end{aligned}$$

Hence we can represent the BER by SNR:

$$\text{BER}=\int_{\sqrt{2SNR}}^{+\infty}\frac{1}{\sqrt{2\pi}}\times\exp\left(-\frac{(\sqrt{2SNR})^2}{2}\right)\text{d}\sqrt{2\text{SNR}} \tag{41}$$

We can simplify the equation (15) by the complementary error function(erfc)[3]:

$$\text{BER}=0.5\,\text{erfc}\left(\sqrt{\text{SNR}}\right)=0.5\,\text{erfc}\left(\sqrt{\frac{Eb}{N_o}}\right) \tag{42}$$

Figure 3 presents the theoretical BER of BPSK modulation under BI-AWGN channel without any coding. Also we simulated the results of randomly transfer 1000000 bits, when $Eb=0,0.5,1,\cdots,9.5,10$ and plotted in the same figure.
As the result shows, the simulation is quite similar to the theoretical BER.

4.3 BER of Different Error Correcting Codes

Before using the Neural Network approaches to solve the problem, we first generated several 'toy' experiments. We found several libraries and tools of matlab and python to help us discover the BER of codes, so we made several experiments on some of the mentioned codes by using these approaches.

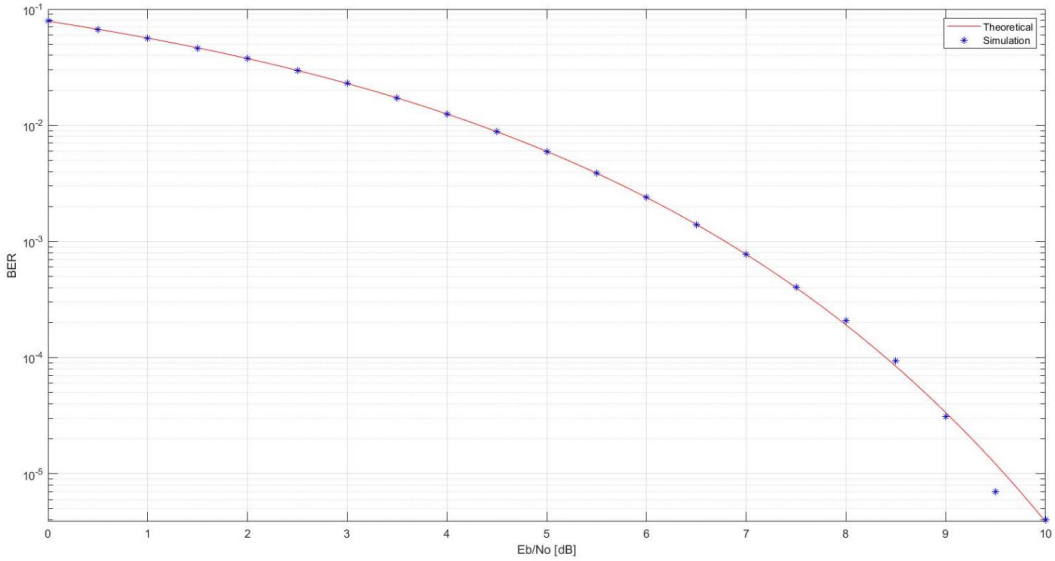

Figure 3: The Theoretical and Simulation Results of BER of BPSK Modulation under BI-AWGN Channel without any coding

4.3.1 Approach 1: bertool

With the help of the 'bertool' in matlab, here we created the plot of BER of several error correcting codes. Simulation of BPSK with Hamming code(7,4), RS code(7,4), convolutional code(7, [171 133]) (both soft and hard decision methods)[4] are shown in figure 4 and BER values are summarized in Table 1.

Table 1: BER of Different Error Correcting Codes

$EbN0(dB)$	Without coding	Convolutional-hard	Convolutional-soft	Hamming	RS
0	0.0786	0.5	0.5	0.0858	0.1154
1	0.0562	0.5	0.0783	0.0598	0.091
2	0.0375	0.5	0.0071	0.0382	0.0662
3	0.0228	0.1246	4.28E-04	0.0218	0.0432
4	0.0125	0.0133	1.74E-05	0.0109	0.0243
5	0.0059	9.92E-04	4.40E-07	0.0045	0.0113
6	0.0023	5.25E-05	5.60E-09	0.0015	0.0041
7	7.72E-4	1.89E-06	2.70E-11	4.09E-04	0.0011
8	1.90E-4	3.99E-08	3.57E-14	7.80E-05	2.28E-04
9	3.36E-5	3.88E-10	8.99E-18	1.00E-05	2.97E-05
10	3.87E-6	1.35E-12	2.74E-22	7.83E-07	2.34E-06

4.3.2 Approach 2: digcommpy

However, the bertool in matlab only includes several kinds of codes. To implement the experiments on the other codes, one needs to use 'berawgn' function with 'bercoding'.

The python library 'digcommpy' provides another convenient solution. It includes several modulation methods, different channels, with various of en/decoders.

Here we implemented the experiment on Polar encoder/decoder, using the required parameters: BI-AWGN channel, BPSK modulation, $k = 64, R = 0.25$. The results are presented in figure 5 and table 2.

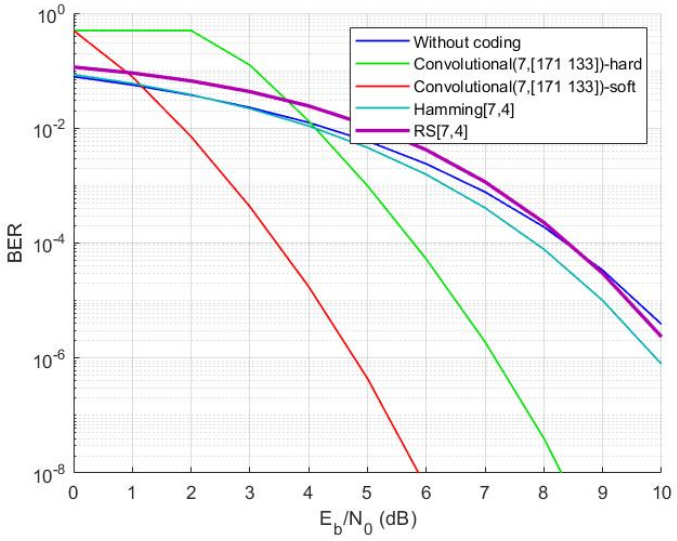

Figure 4: BER of Different Error Correcting Codes

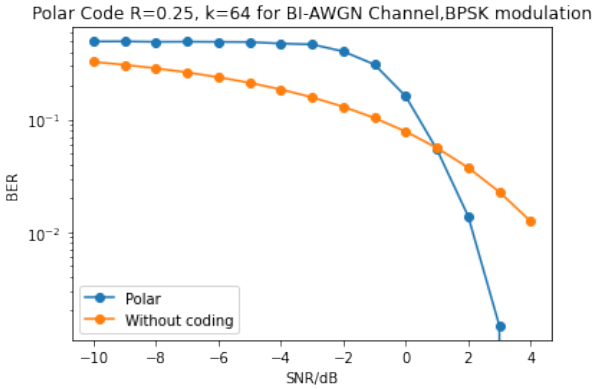

Figure 5: Polar Code R=0.25,k=64 for BI-AWGN Channel, BPSK modulation

Table 2: BER of Polar Encoder/Decoder

$EbN0(dB)$	Without coding	Polar
-10	0.3273	0.497
-9	0.3079	0.4977
-8	0.2867	0.4931
-7	0.2637	0.4945
-6	0.2392	0.4928
-5	0.2132	0.4912
-4	0.1861	0.4763
-3	0.1583	0.4680
-2	0.1306	0.4053
-1	0.1037	0.3100
0	0.0786	0.1616
1	0.0562	0.0543
2	0.0375	0.0139
3	0.0228	0.0015
4	0.0125	0.0

5 Neural Network Approach

5.0.1 Neural Network Method Introduction and Simulation of the performance for the following BI-AWGN channel, BPSK modulation

In this section, we will look at the NN implementation of project problem statement.
We have used following Neural Network structure shown in figure below:

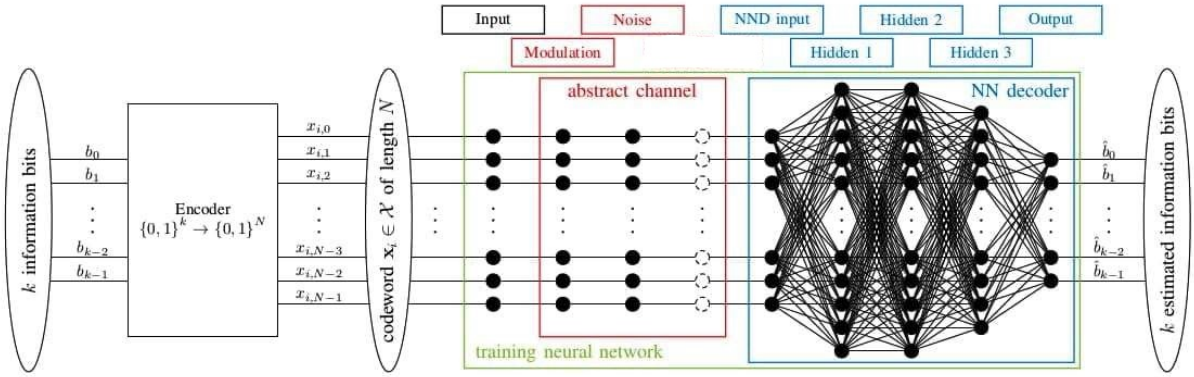

Figure 6: Neural Network setup for channel decoding

We used NN for decoding of noisy codewords. At the transmitter, k information bits are encoded into a codeword of length N . The coded bits are modulated and transmitted over a noisy channel. At the receiver, a noisy version of the codeword is received and the task of the decoder is to recover the corresponding information bits. For channel and modulation we used, binary phase shift keying (BPSK) modulation and an additive white Gaussian noise (AWGN) channel [1].
A NN consists of many connected neurons. In such a neuron all of its weighted inputs are added up, a bias is optionally added, and the result is propagated through a nonlinear activation function which we have used during our implementation, e.g., a sigmoid function or a rectified linear unit (ReLU), defined as follows:

$$g_{sigmoid}(z) = \frac{1}{1+e^{-z}} \qquad g_{relu}(z) = max\{0, z\} \tag{43}$$

For NN parameters we used:

- NN Structure = [N,128, 64, 32,k] where N nodes input layer and [128,64,32] hidden layers, and k nodes output layer
- k = 8 which is number of information bits
- N = 4*k code lengths where rate: r=k/N
- Batch size = 256 which is size of batches for calculation the gradient
- Epoch = 2^{power} where power is introduced to generate tests and saving output results as epoch $2^{power}.txt$ and compare them in the end. We will have figure based on epoches comparison in results section.

We introduced some necessary function for our calculations. Moreover, for optimization we used 'adam' which s a replacement optimization algorithm for stochastic gradient descent for training deep learning models, as for metric we used our own generated function Bit Error Rate(BER), and finally for loss function we used mean squared error (MSE):

$$L_{mse} = \frac{1}{k}\sum_i (b_i - \hat{b_i})^2 \tag{44}$$

5.1 Data Generation and Neural Network Training/Testing

For generation of data, we used random number for creating set of all possible code words generator with all possible information words. After we fit and train NN, our model summary shows that we have 14,824 trainable parameters include in our model. In testing part: We defined test parameters as test batch size, number of codewords, SNR starting point of E_b and ending point of E_b with SNR points.

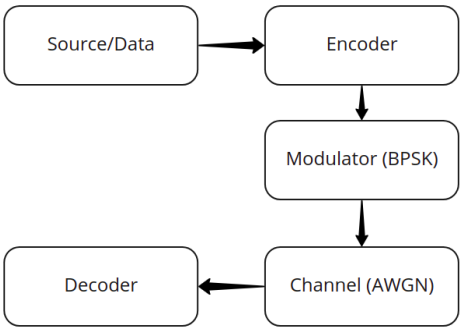

Figure 7: Experiment structure

5.2 Plot the results as curves of an bit error rate BER in dependence of E_b/N_0

We have demonstrated results as Neural Network performance curve which is energy per bit to noise spectral density ratio vs Bit Error Rate. In this subsection we see performance curve for epoch power=10, meaning in 5.0.1 section we introduced parameters where in this curve we have epoch 2^{10}, illustrated in figure 8.

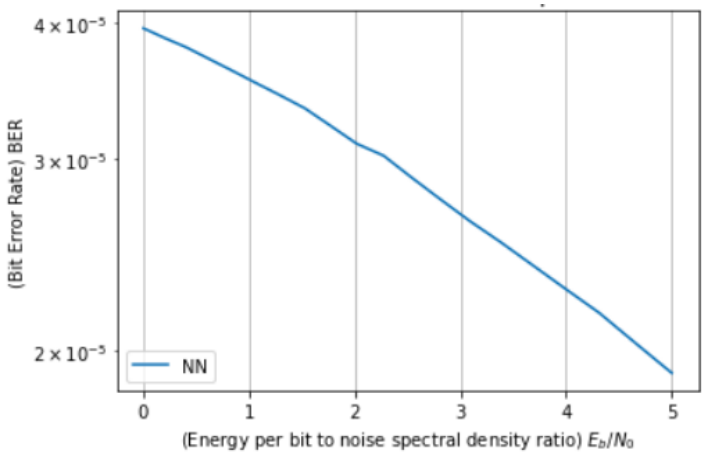

Figure 8: Neural Network Performance Curve when epoch=2^{10}

5.3 Bit-Error-Rate Plot Analysis on Different Epoches

Above we have seen result for one case while in this subsection, we have experimented bit-error-rate for different epoches. The idea as follows we manually set different epoches in paramters section by changing power variable in our case we have performed 5 cases $2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}$ and we recorded data of nb errors to .txt which is provided in our source code directory where you can plot and you can also try your more experiment for different cases using our code. Based on our experiments we got following result, illustrated in figure 9.

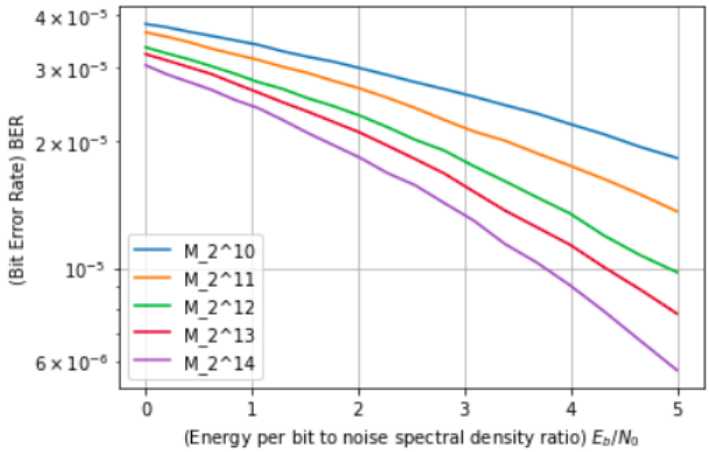

Figure 9: Neural Network Performance Curve on 5 Different Epoches

5.4 Neural Network Results Comparison with Repetition Codes

Further on the experiments, we have compared our results with repetition codes when n=3 and rate R=1/3 then encoder hold following equations: if m=0, c=000, s=[+1+1+1]. if m=1, c=111, s=[-1,-1,-1]

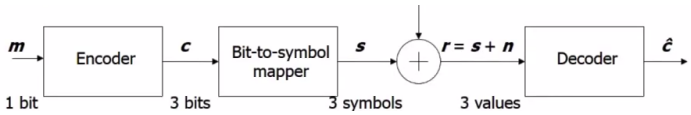

Figure 10: Repetition Codes Communication Channel

We have implemented soft decision and hard decision decoding methods for comparison with neural network in one single performance curve. The first decoding is Hard Decision Decoding which hold following: if $r_i > 0, b_i = 0$ if $r_i < 0, b_i = 1$ for Hard Decision Decoding we use following figure illustrate:

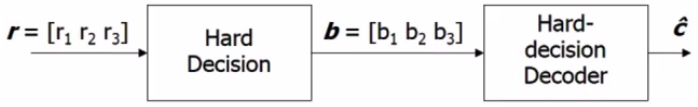

Figure 11: Hard Decision Decoding Scheme

The second decoder we have seen was Soft Decision Decoder that hold following equation: if $|r - [+1 + 1 + 1]|^2 < |r - [-1 - 1 - 1]|^2$ or $r_1 + r_2 + r_3 > 0$ then $\hat{c} = 000$ else $\hat{c} = 111$ we can see Soft Decision Decoder scheme below:

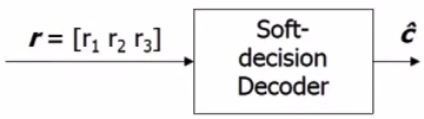

Figure 12: Soft Decision Decoding Scheme

Finally, we can compare the results of the Soft Decision Decoder, Hard Decision Decoding, and Neural Network performance curve in illustration below:

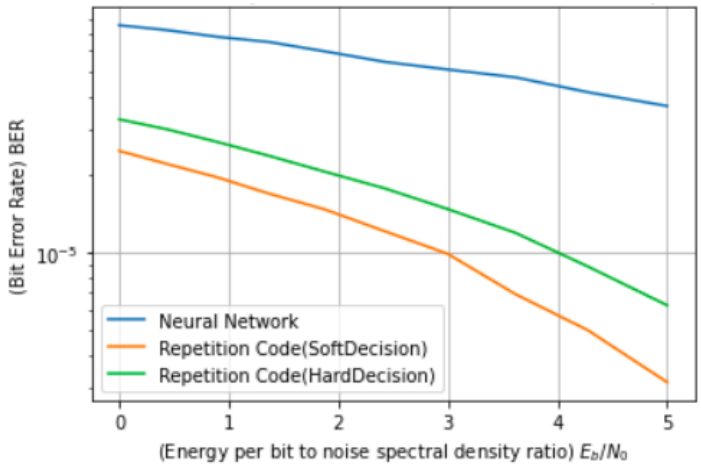

Figure 13: Soft/Hard Decision Decoding and Neural Network Performance Curve when $Epoch = 2^{10}$

6 Conclusion

In this project work, we have learnt how encoding and decoding functions works and studied different methods and approaches to propose a method for the current problem statement.

In the preliminaries section, we first reviewed the definitions of BPSK modulation and BI-AWGN channel. Then we summerized all the error correcting codes we learned during this course. Later on, we deducted the bit error rate of BPSK in BI-AWGN Channel without coding and simulated it. The simulation result is quite close to the theoretical value. Then we used both matlab and python to implement our 'toy' experiments. We used 'bertool' to compare the BER of Hamming code(7,4), RS code(7,4), convolutional code(7, [171 133]) (both softand hard decision methods), it shows that Hamming code and RS code are close to the theoretical curve.Finally with the help of 'digcommpy' library, we implemented the BER calculation on Polar Code. It shows that when SNR/dB less than 1, Polar code has higher BER than theoretical, then it drops shapely to zero when SNR/dB is around 4.

Our NN implementation done based on random codes generation and BPSK modulation as well as AWGN Channel. Our initial experiments were on single epoch which was 2^{10} and we compared in Energy per bit to noise spectral density ratio vs Bit Error Rate perforamnce curve. Moving on, we did experiment with varying epochs our outcomes had shown that as we increase the number of epochs, performance of the neural network curve becomes betters, we saved those results by each training of NN and plotted the performance curve. Moreover, we compared results of NN with repetition codes,and we have discovered that between three cases namely, soft decision decoding, hard decision decoding, and neural network the performance was better on the soft decision decoding among them which was illustrated in figure 13. We studied and updated our results until we achieve certain results and made our conclusions.

References

[1] T. Gruber, S. Cammerer, J. Hoydis and S. t. Brink, "On deep learning-based channel decoding," 2017 51st Annual Conference on Information Sciences and Systems (CISS), Baltimore, MD, USA, 2017, pp. 1-6, doi: 10.1109/CISS.2017.7926071.

[2] L. Huang, H. Zhang, R. Li, Y. Ge and J. Wang, "AI Coding: Learning to Construct Error Correction Codes," in IEEE Transactions on Communications, vol. 68, no. 1, pp. 26-39, Jan. 2020, doi: 10.1109/TCOMM.2019.2951403.

[3] V. Pushpa, H. Ranganathan and M. Palanivelan "BER Analysis of BPSK for Block Codes and Convolution Codes Over AWGN Channel" International Journal of Pure and Applied Mathematics, Volume 114 No. 11 2017, 221-230

[4] J. Van Wonterghem, A. Alloum, J. J. Boutros and M. Moeneclaey, "Performance comparison of short-length error-correcting codes," 2016 Symposium on Communications and Vehicular Technologies (SCVT), Mons, Belgium, 2016, pp. 1-6, doi: 10.1109/SCVT.2016.7797660.

[5] A. Haroon, F. Hussain, and M. R. Bajwa, 'Decoding of Error Correcting codes Using Neural Networks', Dissertation, 2013.

[6] A. Bennatan, Y. Choukroun and P. Kisilev, "Deep Learning for Decoding of Linear Codes - A Syndrome-Based Approach," 2018 IEEE International Symposium on Information Theory (ISIT), Vail, CO, USA, 2018, pp. 1595-1599, doi: 10.1109/ISIT.2018.8437530.

Appendix: Team members' contributions

Kodirjon Akhmedov

1. Reviewing literature on the topic
2. Implement the NN part
3. Preparing the report
4. Preparing the plots
5. Preparing the presentation

Meixing Liao

1. Reviewing literature on the topic
2. Implement the non-NN part
3. Preparing the report
4. Preparing the plots
5. Preparing the presentation