Sequential neural network decoder for convolutional code with large block sizes

Xianhua Yu
Iowa State University

Sequential neural network decoder for convolutional code with large block sizes

by

Xianhua Yu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Electrical Engineering (Communications and Signal Processing)

Program of Study Committee:
Zhengdao Wang, Major Professor
Aleksandar Dogandžić
Jiming Song

Iowa State University

Ames, Iowa

2020

DEDICATION

I would like to dedicate this thesis to my dad, to my mom and to my girlfriend without whose support I would not have been able to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BER : Bit error rate

BPSK : Binary Phase Shift Keying

E_b/N_0 : Energy per bit to noise power spectral density ratio

GRU: Gated recurrent unit

QPSK : Quadrature Phase Shift Keying

QAM: Quadrature amplitude modulation

RNN : Recurrent neural network

SNR : Signal to noise ratio

SNND : Sequential neural network decoder

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Zhengdao Wang for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Aleksandar Dogandžić and Dr. Jiming Song. I would additionally like to thank my group partner Qi Xiao, Pan Zhong, Yimeng Li and Chenyu Xu for their company during the hard time. I would also like to thank my friends and family for their loving guidance and financial assistance during the writing of this work.

ABSTRACT

Due to the curse of dimensionality, the training complexity of the neural network based channel-code decoder increases exponentially along with the code word's length. Although computation power has made significant progress, it is still hard to deal with long block length code word. In this thesis, we proposed a neural network based decoder termed as Sequential Neural Network Decoder (SNND). The SNND consists of multiple sub models, and it passes the last state of the current sub model to the following model as the initial state. The bit error rate (BER) performance of the SNND remains unchanged during the number of sub models increases, it achieves a performance closes to the performance of Viterbi soft decision under Additive white Gaussian noise (AWGN) channel. However, the SNND's performance is found to decrease along with the modulation order increase.

CHAPTER 1. OVERVIEW

In this chapter, we will give an overview of this thesis, which consists of introduction, motivation, and outline.

1.1 Introduction

Deep learning has achieved tremendous success in the field of speech processing and computer version which had led to the exploration of deep learning in the field of communication recently. The potential application for deep learning in communication includes neural network based channel decoder, neural network based multiple-input multiple-out (MIMO) detector, even in learning the communication system.

Compared with the traditional channel code decoding method, the performance of the neural network-based channel decoder is close to the conventional method and has the advantages of low-latency and non-iterative. Nevertheless, the neural network based channel decoder has the limitation in learning the long block lengths code word which is known as the curse of dimensionality. The challenge of the curse of dimensionality was proposed by Wang and Wicker (1996) which lead exponentially complexity during the training process. The curse can be explained as for k input information bits, the neural network needs to distinguish between 2^k possible code words. In order to learn the full code book, it will result in exponential training complexity during the training process.

The number of possible outputs is limited in deep learning such as computer vision. On the contrary, the training set with an unlimited number of labels is available in neural network based channel code decoder because we already known the channel model and the encoding process. Although a lot of powerful deep learning libraries like tensorflow and pytorch are widely used right now and the computer's computation power had made significant progress, the exponential complexity still impedes learning the long block length code words by the neural network, which was shown in Gruber et al. (2017). In Cammerer et al. (2017), some insight was offered to solve this challenge.

The convolutional codes are a type of error-correcting codes that generate code word by transmitting information sequence through a linear finite-state shift register. Because of the good performance of the convolutional codes, it has been widely used in communication systems such as Global System for Mobile Communications (GSM), Wideband Code Division Multiple Access (WCMDA), Code-division multiple access (CDMA) 2000 and IEEE 802.11.

In this thesis, we propose a gated recurrent unit (GRU) based model to deal with the curse of dimensionality problem, which is termed as the Sequential Neural Network Decoder (SNND). Due to the curse, it is hard to learn long block length code word straightforwardly. An intuitively idea is truncate code word to multiple parts with small size and each part assign to a sub model. We will take this approach and use the GRU sub model, which concatenates three layers of GRU network and one fully connected layer. The SNND consists of multiple sub GRU models and will pass the last state of the current sub GRU model to the following GRU model as the initial state.

According to the numerical results, the SNND achieves an excellent performance compared with the Viterbi soft decision, and the performance of the SNND remains unchanged as the number of sub GRU models increases. Due to the shift register, the adjacent convolutional code words are correlated that affect each other in the decoding process. The process of passing the last state can keep the integrity of the training noisy code word that enables each sub GRU model to learn the convolutional code thoroughly. Therefore, the amount of sub GRU models increase will not affect the SNND's performance. However, the SNND's performance will decrease with the modulation order increase because the training complexity increases.

1.2 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 reviews the past works and literature related to our work.

Chapter 3 provides the background information for our thesis which includes the convilutional code, the Viterbi decoding algorithm and the recurrent neural network.

Chapter 4 Presents the problem formulation, the system model and our proposed models. The proposed models consists of GRU model and SNND.

Chapter 5 Reports numerical result and discusses the reason behind the results.

Chapter 6 provides a summary of this thesis and lays out the future research directions.

CHAPTER 2. LITERATURE REVIEW

The work to understand why the primary cell in the human brain connected together can produce a high complexity pattern is McCulloch and Pitts (1943), and the authors used a simple model to illustrate it. McCulloch and Pitts (1943) made a considerable contribution to further research in artificial neural networks, which gave insight into solving problems using features of the human brain's neurons. An inchoate type of neural network was proposed by Hopfield (1982), which is called Hopfield network. The Hopfield network's key point is similar to the maximum likelihood in channel code decoding. It will let the error code word converge to the nearest state that is the most likely code word. The Hopfield network can converge to a local minimum but sometimes the local minimum is a wrong local minimum.

The feed forward neural network displaced the Hopfield network in short time because the feed forward neural network can learn the mapping between input information with noise and output label. There is no need to assume the noise as before; the feed forward neural network is able to learn the mapping directly between noisy input and output.

Due to the neural network's ability, people started to think about whether to use neural networks to decode channel code. Tallini and Cull (1974) showed a way to decode Hamming code by using syndrome as the input of neural network. An artificial neural network Viterbi decoder was proposed by Wang and Wicker (1996), whose decoding process is much faster than the digital Viterbi decoder. But both of them mentioned the shortcomings of the neural network; in the process of decoding, the possible correct decoding path of the neural network is much more than the standard decoder. Because the shortcomings in the neural network and the computer was not powerful like today, they only could decode short block length code words in their work. Later, a new neural approach using recurrent neural network was proposed by Hamalainen and Henriksson (1999), which showed the possibility of using the recurrent neural network to decode long constraint length convolutional code.

The research in decoding channel code by the neural network only made little progress because the training techniques can not build a neural network with numerous neurons and layers. This situation did not change until Hinton et al. (2006) proposed a greedy learning algorithm that made training multiple layers neural network possible. As time flies, researchers has made incredible progress in hardware field, like graphical processing units. As a result of the progress, we can not only build a deep neural network consisting of a lot of layers and neurons but also reduce training time. Therefore, further research in decoding channel code by using the neural network becomes accessible.

Gruber et al. (2017) showed using the neural network to learn the structured code is easier than to learn random code. Also, they address the challenge that it is hard to learn long-block length codes because the computation complexity will increase exponentially as the number of information bits increases. Therefore, Cammerer et al. (2017) proposed a partitioned successive cancellation list (PSCL) decoder, which decodes long block length Polar code by dividing code word into sub-block and each block corresponds to one sub neural network model. But the partitioned successive cancellation list decoder had a considerable drawback, for a fixed size sub neural decoder, the performance will be decrease as the number of sub neural decoders increases.

In order to find the most suitable neural network architecture for channel code decoding, Lyu et al. (2018) provide a experiment between the multi-layer perceptron (MLP), convolution neural network (CNN) and recurrent neural network (RNN). The numerical results showed the recurrent neural network achieve the performance which means RNN is a suitable architecture for channel code decoding. According to results, the research in channel decoding by neural network can do further. Tandler et al. (2019) proposed a neural decoder consisting of gated recurrent unit (GRU) layers such that the performance of decode the convolutional noisy code word with constraint length $m = 6$ is almost the same as that of the Viterbi soft decision. During their training process, they used a method called prior ramp-up from the filed of natural language and speech processing to enhance the neural decoder's performance.

Some researches are in a new direction that prefers to use neural networks to create a code than learn channel code. In Jiang et al. (2020) and Ye et al. (2019), they proposed an auto-encoder-based neural network that can achieve almost and even better performance than conventional decoder.

CHAPTER 3. BACKGROUND

In this chapter, we will introduce the background related to our research. Our primary research purpose is using neural networks to decode convolutional code so that we will present the theory of convolutional code and different decoding methods. The decoding methods will include the Viterbi decoding algorithm and the neural decoder.

3.1 Convolutional Codes

Convolutional codes were first proposed by Elias (1955) as an improved version for the block code. The process of generate convolutional code can be explained as transmit the information sequence through a linear finite-state shift register. A nonsystematic convoltional encoder with rate $R = 1/2$ and memory order $m = 3$ is shown in figure 3.1. The encoder consists of $m = 3$ delay elements and $n = 2$ modulo-2 adders. The encoder is a linear system because the modulo-2 addition is a linear operation.

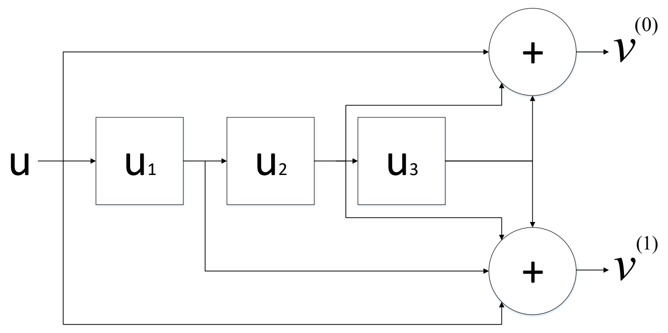


Figure 3.1 $(2, 1, 3)$ convolutional encoder

The bits $u = (u_0, u_1, u_2...)$ enters the encoder one bit for once. The sequence u convolution with two impulse response of the encoder will result in two encoder output sequence $v^{(0)} = (v_0^{(0)}, v_1^{(0)}, v_2^{(0)}...)$ and $v^{(1)} = (v_0^{(1)}, v_1^{(1)}, v_2^{(1)}...)$. The two impulse responses can be denoted as

$g^{(0)} = (g_0^{(0)}, g_1^{(0)} ..., g_m^{(0)})$ and $g^{(1)} = (g_0^{(1)}, g_1^{(1)} ..., g_m^{(1)})$ with memory order m of an encoder. For figure 3.1, the upper adder is connected to information sequence, stages 2 and stages 3, so $g^{(0)} = (1\ 0\ 1\ 1)$. The lower adder is connected to information sequence, stages 1, stages 2 and stages 3, hence $g^{(0)} = (1\ 1\ 1\ 1)$. The impulse response also named generated sequence of the encoder.

The process of generating convolutional code is shown as follow. The information bits u input to the encoder, the two output sequence can be expressed as

$$v^{(0)} = u \circledast g^{(0)} \tag{3.1}$$

$$v^{(1)} = u \circledast g^{(1)} \tag{3.2}$$

where $\circledast$ denotes the operation of discrete convolution. The output code sequence v will combine $v^{(0)}$ and $v^{(1)}$ as

$$v = (v_0^0 v_0^1, v_1^0 v_1^1, v_2^0 v_2^1, ...) \tag{3.3}$$

The operation of convolution in time domain is equivalent to polynomial multiplication in transform domain. We define the D transform of v as

$$v(D) = \sum_{i=0}^{\infty} v_i D^i \tag{3.4}$$

thus encoding equation is rewritten as

$$v^{(0)}(D) = u(D)g^{(0)}(D) \tag{3.5}$$

$$v^{(1)}(D) = u(D)g^{(1)}(D) \tag{3.6}$$

The information bits $u(D)$, the coded sequence $v^{(0)}(D)$ $v^{(1)}(D)$, and the generated sequence $g^{(0)}(D)$ $g^{(1)}(D)$ can be rewritten as

$$u(D) = u_0 + u_1 D + u_2 D^2 + ... \tag{3.7}$$

$$v^{(0)}(D) = v_0^{(0)} + v_1^{(0)} D + v_2^{(0)} D^2 + \cdots \tag{3.8}$$

$$v^{(1)}(D) = v_0^{(1)} + v_1^{(1)} D + v_2^{(1)} D^2 + \cdots \tag{3.9}$$

$$g^{(0)}(D) = g_0^{(0)} + g_1^{(0)} D + g_2^{(0)} D^2 + \cdots \tag{3.10}$$

$$g^{(1)}(D) = g_0^{(1)} + g_1^{(1)} D + g_2^{(1)} D^2 + \cdots \tag{3.11}$$

Hence, the code word can be rewritten as

$$V(D) = [v^0(D), v^1(D)] \tag{3.12}$$

$$v(D) = v^{(0)}(D^2) + Dv^{(1)}(D^2) \tag{3.13}$$

The $v(D)$ showed above is a special case for $n = 2$, we want to extend it to general case for an encoder with arbitrary k-input and n output. Because encoder is a linear system, k-input and n output will results numbers of kn transfer function. The transfer function can be represented by the generator matrix shows below

$$G(D) = \begin{bmatrix} g_1^{(0)}(D) & g_1^{(1)}(D) & \cdots & g_1^{(n-1)}(D) \\ g_2^{(0)}(D) & g_2^{(1)}(D) & \cdots & g_2^{(n-1)}(D) \\ \vdots & \vdots & \vdots & \\ g_k^{(0)}(D) & g_k^{(1)}(D) & \cdots & g_k^{(n-1)}(D) \end{bmatrix} \tag{3.14}$$

with generator matrix,the (n, k, v) convolutional encoder's coding equation be represented as

$$V(D) = U(D)G(D) \tag{3.15}$$

$$v(D) = v^{(0)}(D^n) + Dv^{(1)}(D^n) + \cdots + D^{(n-1)}v^{(n-1)}(D^n) \tag{3.16}$$

where the input sequence $U(D) = [u^{(1)}(D), u^{(2)}(D), \cdots, u^{(k)}(D)]$, the output sequence $V(D) = [v^{(1)}(D), v^{(2)}(D), \cdots, v^{(k)}(D)]$, and $v(D)$ is the code word. Combining equations 3.14, 3.15, 3.16, the code word $v(D)$ can be rewritten as

$$v((D) = \sum_{l=1}^{k} u^{(i)}(D^n)g_i(D) \tag{3.17}$$

where composite generator polynomial $g_i(D) = g_i^{(0)}(D^n) + Dg_i^{(1)}(D^n) + \cdots + D^{(n-1)}g_i^{(n-1)}(D^n)$.

Let us use a simple example to illustrate equation 3.17. Assuming the information bits $u = (1\ 0\ 1\ 1\ 1)$ enters encoder in figure 3.1. From previous statement, we can easily obtain information sequence $u(D) = 1 + D^2 + D^3 + D^4$, and the generate polynomial $g^{(0)}(D) = 1 + D^2 + D^3$, $g^{(1)}(D) = 1 + D^2 + D^3 + D^4$. Hence, the composite generator polynomial is expressed as

$$g(D) = g^{(0)}(D^2) + Dg^{(1)}(D^2) = 1 + D + D^3 + D^4 + D^5 + D^6 + D^7 \tag{3.18}$$

The calculation used GF (2). Then the code word is

$$\begin{aligned} v(D) &= u(D^2)g(D) \\ &= (1+D^4+D^6+D^8)(1+D+D^3+D^4+D^5+D^6+D^7) \\ &= 1+D+D^3+D^7+D^9+D^{11}+D^{14}+D^{15} \end{aligned} \tag{3.19}$$

so the code word $v = (1\ 1, 0\ 1, 0\ 0, 0\ 1, 0\ 1, 0\ 1, 0\ 0, 1\ 1)$.

3.2 Viterbi Decoding Algorithm

The Viterbi algorithm proposed by Viterbi (1967), is the optimum decoding algorithm for the convolutional code. Many types of research for finding optimal decoding algorithm was done before Viterbi. The earliest one was proposed by Wozencraft (1957), which is using a sequential decoding to decode convolutional code with long constraint length. The sequential decoding method was modified by FANO (1963). After Viterbi, Omura (1969) showed how to find the shortest path through a weight graph by dynamic programming is equivalent to the Viterbi algorithm. Then Forney (1973) figured out the Viterbi algorithm is a maximum likelihood algorithm which means the decoder will always choose the decoded bit sequence to maximize the conditional probability of the received sequence. The soft-output Viterbi algorithm was proposed by Hagenauer and Hoeher (1989) as an alternative version of the Viterbi algorithm.

For an (n, k, v) convolutional encoder and an information sequence u of length $K = kh$ (h is the length of input sequence), there are 2^k branches getting in and out each states, and 2^K unique paths in trellis. Assuming the information sequence $u = (u_0, u_1, \cdots, u_{h-1})$ of length K enters the (n, k, v) encoder results in the code word $v = (v_0, v_1, \cdots, v_{h+m-1})$ of length $L = n(h+m)$. After code word v pass through discrete memory-less channel (DMC), we will receive sequence $y = (y_0, y_1, \cdots, y_{h+m-1})$. We can use a simple notation to rewrite u, v, y, which $u = (u_0, u_1, \cdots, u_K)$, $v = (v_0, v_1, \cdots, v_L)$, and $y = (y_0, y_1, \cdots, y_L)$. The convolutional decoder should output a sequence $\hat{v}$, which is estimation of code word v. And the estimation is based on the received sequence y. The convolutional decoder is a maximum likelihood (ML) decoder that we will maximize the log-likelihood function $\log P(\mathbf{y}|\mathbf{v})$ for select $\hat{v}$ as v under channel of DMC. For channel of DMC

$$P(\mathbf{y}|\mathbf{v}) = \prod_{l=0}^{h+m-1} P(\mathbf{y}_l|\mathbf{v}_l) = \prod_{l=0}^{L-1} P(y_l|v_l) \tag{3.20}$$

the log-likelihood is given by

$$\log \ P(\mathbf{y}|\mathbf{v}) = \sum_{l=0}^{h+m-1} \log \ P(\mathbf{y}_l|\mathbf{v}_l) = \sum_{l=0}^{L-1} \log \ P(y_l|v_l) \tag{3.21}$$

where $P(y_l|v_l)$ is the channel transition probability, $\log(\mathbf{y}|\mathbf{v})$ is log-likelihood function can be denoted as $M(\mathbf{y}|\mathbf{v})$ which also named path metric. The branch metrics is $\log \ P(\mathbf{y}_l|\mathbf{v}_l)$ in equation 3.21, which is denoted by $M(\mathbf{y}|\mathbf{v})$. And $log \ P(y_l|v_l)$ is called bit metrics, which is denoted by $M(y_l|v_l)$. So that the path metric $M(\mathbf{y}|\mathbf{v})$ can be rewritten as

$$M(\mathbf{y}|\mathbf{v}) = \sum_{l=0}^{h+m-1} M(\mathbf{y}|\mathbf{v}) = \sum_{l=0}^{h+m-1} log \ P(\mathbf{y}_l|\mathbf{v}_l) = \sum_{l=0}^{L-1} M(y_l|v_l) = \sum_{l=0}^{L-1} log \ P(y_l|v_l) \tag{3.22}$$

And the first t branches of the path can be expressed by path metric as follow

$$M([\mathbf{y}|\mathbf{v}]_t) = \sum_{l=0}^{t-1} M(\mathbf{y}|\mathbf{v}) = \sum_{l=0}^{t-1} log \ P(\mathbf{y}_l|\mathbf{v}_l) = \sum_{l=0}^{nt-1} M(y_l|v_l) = \sum_{l=0}^{nt-1} log \ P(y_l|v_l) \tag{3.23}$$

Now, we cite the whole process of the Viterbi algorithm from Lin and Costello (2001)

Algorithm 1: The Viterbi algorithms

```
Step 1. Beginning at time unit $t = m$, compute the partial metric for the single path
    entering each state. Store the path (the survivor) and its metric for each state.
Step 2. Increase $t$ by 1. Compute the partial metric for all $2^k$ paths entering a state by
    adding the branch metric entering that state to the metric of the connecting survivor at
    the previous time unit. For each state, compare the metrics of all $2^k$ paths entering that
    state, select the path with the largest metric (the survivor), store it along with its metric,
    and eliminate all other paths.
Step 3. If $t < h + m$, repeat step 2; otherwise, stop.
```

The previous algorithm is used to find the maximum likelihood path through the trellis with the largest metric for the received y through the channel. The computation in the Viterbi algorithm is operation of add, compare and select in step 2. At each time step it adds 2^k branch metrics to each previously stored path metric, it compares each state's 2^k paths metrics, and it select the survivor, which is the largest metric though the path. The algorithm will start at all-zero state and end with all-zero state, so there is only one survivor.

Theorem 1 *The final survivor $\hat{y}$ in the Viterbi algorithm is the maximum likelihood path; that is,*

$$M(\boldsymbol{r}|\hat{\boldsymbol{y}}) \geq M(\boldsymbol{r}|\boldsymbol{y}), \ \textit{all} \ \boldsymbol{y} \neq \hat{\boldsymbol{y}} \tag{3.24}$$

A proof of this theorem can be found in Lin and Costello (2001) in the book.

The performance bounds for convolutional codes, and the details can be found at Lin and Costello (2001) chapter 12.

3.3 Recurrent Neural Network

The basis of recurrent neural network (RNN) is Rumelhart et al. (1986).The Recurrent neural network is a class of artificial neural network which is widely used in language modeling.

3.3.1 Vanilla Recurrent Neural Networks

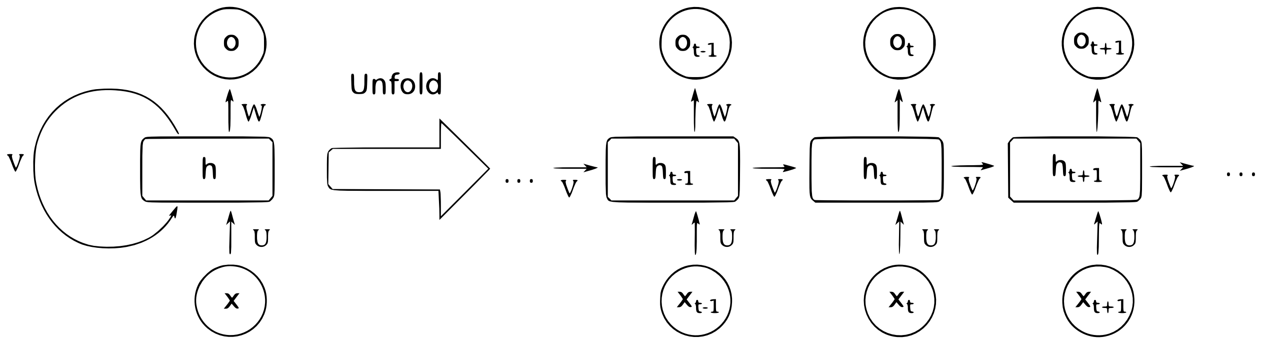


Figure 3.2 Recurrent Neural Network[1]

RNN is recurrent because it performs same function for same input, and the output of the current input depends on the calculation in the past. The RNN can deal with the input sequence

[1]https://en.wikipedia.org/wiki/Recurrent-neural-network/

using their internal state, unlike the feed-forward neural network. The unfolded structure of RNN is shown in figure 3.2 and the mathematical expression is shown in below.

$$h_t = \tanh(W_h x_t + U_h h_{t-1} + b_h) \tag{3.25}$$

$$o_t = W_o h_t + b_o \tag{3.26}$$

where h_t is the hidden layer vector, b is the bias vector and the W is the weight metrics. Due to the recurrent procedure, RNN will cause the vanishing gradient and exploding gradient problem.

3.3.2 Gated Recurrent Units

Gated Recurrent Units(GRU) was introduced by Cho et al. (2014), which are a gating mechanism in recurrent neural networks(RNN). GRU aims to solve the problem which exists in RNN, the vanishing gradient problem. GRU can also be considered as a different type of Long short-term memory (LSTM) because both are designed similarly and perform equally good performance in some cases.

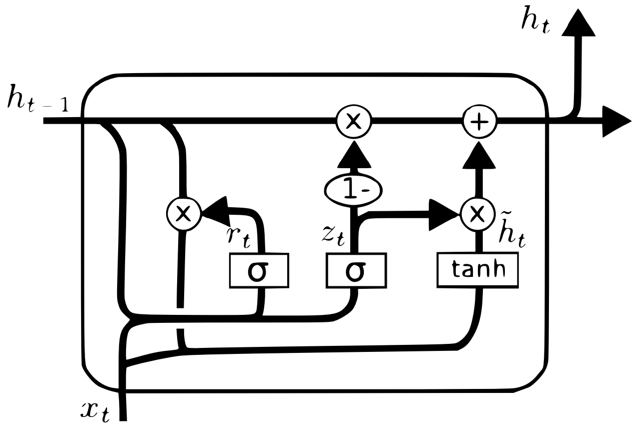


Figure 3.3 Gated recurrent units[2]

Structure of GRU cell, is shown in figure 3.3. According to the notation provided by figure 3.3, the mathematical expressions of the GRU cell shows as follow:

[2]https://technopremium.com/blog/rnn-talking-about-gated-recurrent-unit/

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{3.27}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{3.28}$$

$$\widetilde{h_t} = \tanh(W x_t + r_t \cdot U h_{t-1}) \tag{3.29}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \widetilde{h_t} \tag{3.30}$$

where notation of $\cdot$ is Hadamard product, σ is sigmoid function, tanh is hyperbolic tangent. r_t is Reset gate vector, z_t is update gate vector, $\widetilde{h_t}$ is current memory content, h_t is final memory at current time step, x_t is input vector and W, U are parameter matrices.

To prevent the vanishing gradient problem in standard RNN, GRU uses a mechanism called update gate and reset gate. Generally speaking, two vectors decide what information should be passed to output or forgot in the process. Using these two gates in the neural network training process can keep information on what we need from a long time ago, without removing irrelevant information to prediction. Using the update gate also simplifies the GRU cell parameters because the update gate is integrated with LSTM cell's forget gate and input gate. As a result, it could save one gating vector and the corresponded parameters.

Thanks to the benefits of GRU mentioned before, it prevents the vanishing gradient problem in standard RNN and fewer parameters but can achieve nearly performance than LSTM. These advantages motivate GRU as a good candidate for creating a simple model that can decode long sequence channel code with a weak GPU.

CHAPTER 4. METHODS AND PROCEDURES

As mentioned in Wang and Wicker (1996) the complexity of learning channel code by using neural network will increase expotentially along with the information bits increase. Although, the research in hardware like graphics processing unit (GPU) has made considerable progress, it still not easy for us to train a long block length code because the limitation of GPU's memory and neural network model size needed to be fixed. In this chapter, we will propose two approaches. There is GRU model, and Sequential neural netword decoder (SNND). The GRU model is only using one model that consists of three bidirectional GRU layers and one dense layer. The SNND consists of multiple GRU models, each model are correlated by the previous sub model's last state as the initial state of next sub model. Unlike approach in Cammerer et al. (2017) , the SNND shows incredible performance even though the number of sub model increase to 32. In this chapter we will discuss the formulation to our problem and our system model. We will brief introduce the method we proposed.

4.1 Problem Formulation

Using neural decoder to decode coded bits can be simply considered as a mapping between neural network's input and output, which expressed as:

$$Y = M(X) \tag{4.1}$$

where X denotes the input of the decoder, M denotes mapping of decoder, and Y denotes the output of the decoder. The decoding mapping is learned from the training sets, which will be stored as weigh parameter in the neural network. We assume $x = [x_1, x_2, ..., x_n]^T$ as input vector for neural network, where n is the input size of the neural network. The mapping can be expressed more specific as

$$y = \sum_{i=1}^{n} w_i r_i + b = w^T r + b \tag{4.2}$$

where $w = [w_1, w_2, ..., w_n]^T$ is a n-dimension weight vector, and the b denotes bias. Passing y through the activation function $f(\cdot)$ will result in activation value. In order to learn complex decoding mapping, we need to use multiple layers with a certain hierarchical structure. The whole decoding mapping of neural network can be expressed as

$$Y = f(X, \theta) = f^{(L-1)}(f^{(L-2)}(...f^{(0)}(X))) \tag{4.3}$$

where θ is the parameter of neural decoder and the L is the numbers of the layers in neural network.

4.2 System Block Diagram

IEEE 802.11 standard (IEEE (2016)) employs convolutional code as it's channel code specification. The IEEE 802.11 standard uses orthogonal frequency-division multiplexing (OFDM) as it's modulation type. So we slight modify OFDM system model as our system model which remove Fast Fourier transform (FFT) and Inverse Fast Fourier Transform (IFFT). Our system block diagram is shown in figure 4.1.

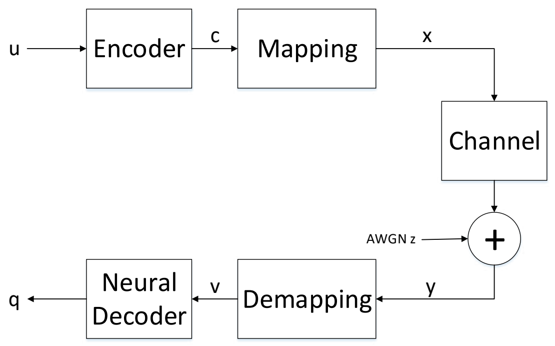


Figure 4.1 System Block Diagram

At transmitter, the input bits u with uniform distribution of length k will be encoded into bits c of length n which correspond to convolutional code definition. Next, coded bits c is mapping to a symbol vector x through constellation mapping. At last, the symbol x will pass through Additive white Gaussian noise (AWGN) channel. The Additive white Gaussian noise denoted by z.

At the receiver, y denotes the symbol we received. Assuming perfect synchronization, y can be expressed as

$$y = Hx + z \tag{4.4}$$

where H is complex channel gain of the symbol and in AWGN channel $H = 1$. Then the received symbol y demapping to bits u with length n through constellation demapping. At last, bits v used for channel decoding with neural decoder, and we used q denotes the neural decoder's output. The input bits are binary, but the output of the neural decoder is floating-point. In order to calculate BER between input bits u and output bits q, we use a small trick that rounds the output bits q to an integer.

4.3 Proposed Models

In this section, we will briefly introduce the method proposed. There are methods as introduction mentioned, GRU model, and Sequential Neural Network Decoder.

4.3.1 GRU Model

Lyu et al. (2018) had showed the empirical result that dealing with the channel code decoding process, RNN performs best BER performance between multilayer perceptron (MLP), CNN (convolutional neural network) and RNN. We reproduce the experiment in Lyu et al. (2018) by adding Long short-term memory (LSTM) and GRU as variation RNN. Our numerical results have shown that RNN performs best between MLP, CNN, and RNN (same as Lyu et al. (2018)), and the LSTM and GRU showed almost equal performance but better than vanilla RNN. Due to the complexity of the GRU unit being lower than the LSTM unit mentioned above, we chose the GRU layer to construct our neural network model.

In Schuster and Paliwal (1997) proposed a new type of RNN called bidirectional recurrent neural network (BRNN). The BRNN training process differs with RNN, which is not only in forwarding time step but also in the backward step, and the performance of BRNN is better than RNN. Inspired by this, we think it is possible to use a bidirectional GRU instead of GRU. After plenty of experiments, we finally select bidirectional GRU to construct our neural network model because bidirectional GRU performance is better than GRU. The GRU model structure is shown in figure 4.2, which consists of three bidirectional GRU layers and one dense layer.

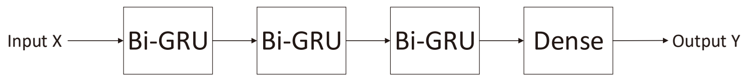
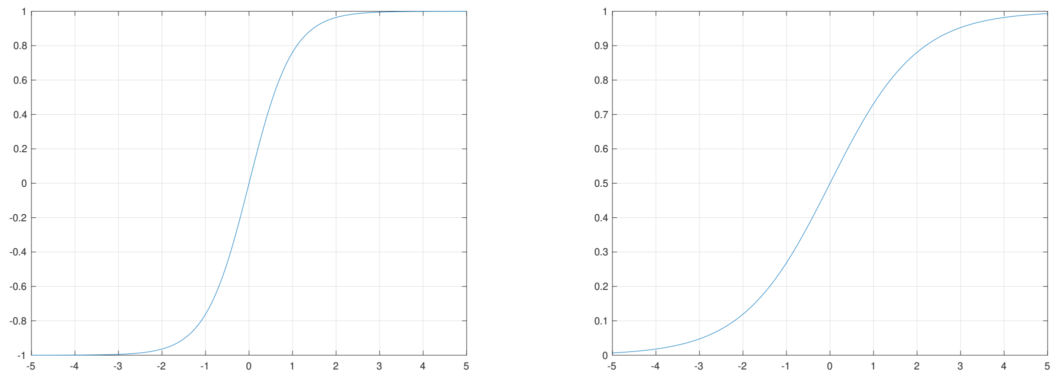


Figure 4.2 GRU model

The GRU layer uses *tanh* as activation function and the dense layer uses *sigmoid* as activation function. And the mathematical expression and the figure of these two activation function is shown as below.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{4.5}$$

$$tanh(x) = \frac{2}{1+e^{-2x}} + 1 \tag{4.6}$$



(a) tanh (b) sigmoid

Figure 4.3 Activation function

The sigmoid activation function is a nonlinear function that translates the input range in $[-\infty, +\infty]$ to the output range in $(0, 1)$. The output range of the sigmoid function is the output range of the decoder that we wished because original uncoded bits are binary. The tanh activation function mathematical expression is like sigmoid, and all of them's curve look like a S-shape.The tanh activation function is also a nonlinear function that translates the input range in $[-\infty, +\infty]$ to the output range in $(-1, 1)$. The tanh function is mainly used classification between two classes, which suitable for our main purpose.

In order to show the GRU model's capacity, we conduct a simple experiment using BPSK modulation and the result shown in figure 4.4. We will evaluate the model's ability by using the method of bit error rate (BER). The bit error rate is defined as the rate at which errors occur in a transmission system. It can be considered as the number of errors that occur in a string of a stated number of bits. The bit error rate mathematical expression is shown as below.

$$BER = \frac{Error\ Bits}{Total\ Number\ of\ Bits} \tag{4.7}$$

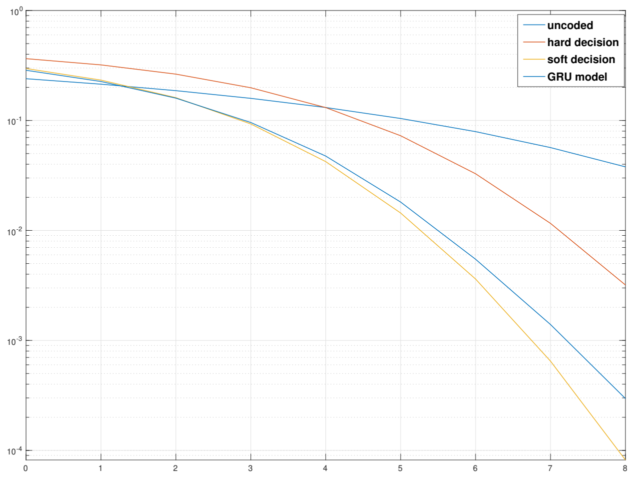


Figure 4.4 block length 100

Figure 4.4 shows BER performance between the GRU model, the uncoded sequence, and the Viterbi decoder. It's easy to see that the GRU model achieves much better performance than Viterbi hard-decision decoding and exists a slight gap with Viterbi soft-decision decoding. It reveals neural network capacity in the channel code decoding process, which is the basis of further research.

To decode a long block length convoltional code, we need to truncate origin noisy code word to a large number of short sequences which the length satisfy GRU model size. Inspried by Cammerer et al. (2017), we denoted this new model as partitioned GRU model. Example is shown in figure 4.5, the decoding step is shown as follow.

1. Truncating input sequence X to n part $x_1, x_2..x_n$,, and the length of each part is the input size of the GRU model.
2. Feeding the data $x_1, x_2...x_n$ to GRU model in turn
3. Concatenating output $y_1, y_2..y_n$ to output Y.

In the following sections, we will continue to use notation X, Y as input and output.

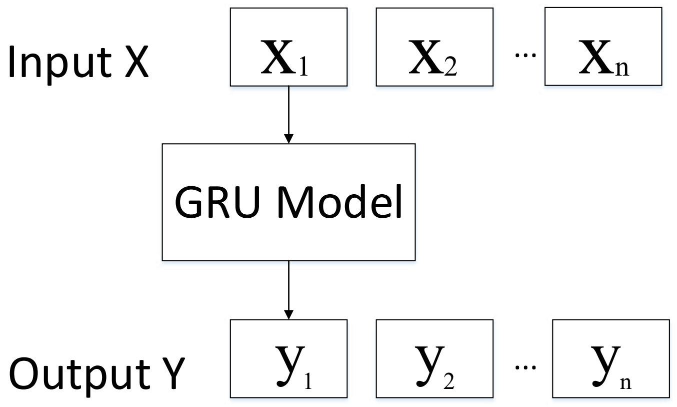


Figure 4.5 GRU mode decoding process

4.3.2 Sequential Neural Network Decoder

Li et al. (2018) proposed a multi-input LSTM architecture. we intuitively think we can create a multiple-input model that each input corresponds to one model. In Cammerer et al. (2017) showed used multiple model to combine to one model that can achieve not bad performance. Zhang et al. (2020) have proposed a decoding process by sharing convolutional code's initial state to achieve better performance. Inspired by them, we intuitively had an idea to address the challenge of decode long block length convolutional code. The idea is to combine multiple GRU models to one model. And due to the new model is sequential, we notated it as Sequential Neural Network Decoder (SNND).

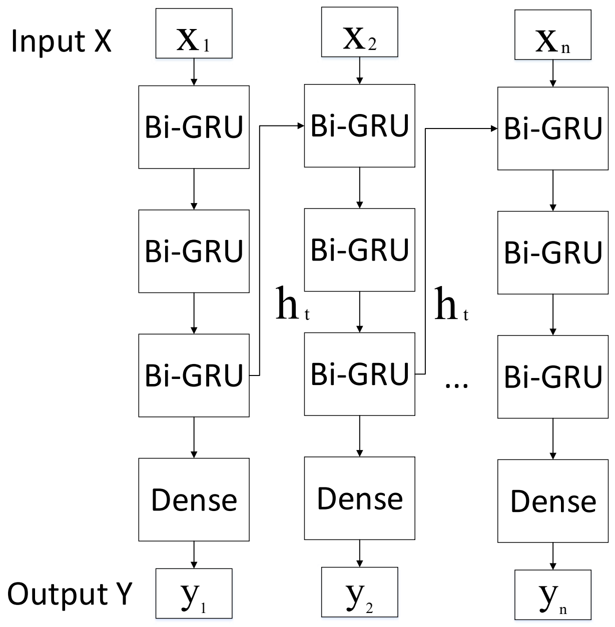


Figure 4.6 Sequential Neural Network Decoder

The SNND structure is shown in figure 4.6, which consists of multiple GRU models. The difference between the SNND and the partitioned GRU model is not only in the number of sub GRU models but the SNND will pass the last state of the current sub GRU model to the following GRU model as the initial state. According to the action of pass state, the SNND can be considered as a continuous model. In order to illustrate the process of pass state more clearly, we will give a mathematical expression as follow. We assume first GRU model's intial state and last state are h_{t11} h_{t13} respectively, and second GRU model's initial state is h_{t21}.

$$h_{t11} = 0 \tag{4.8}$$

$$h_{t21} = z_t \cdot h_{t13} + (1 - z_t) \cdot \widetilde{h_t} \tag{4.9}$$

The step of the decoding process is shown below:

1. Truncating the input noisy convolutional coded sequence X to n parts $x_1, x_2 ... x_n$, and the length of each part is the input size of the sub GRU model.
2. Feeding x_1 to sub GRU model 1 that results in the state h_{t1} and the output y_1.
3. Feeding x_2 and h_{t1} to sub GRU model 2 that results in the state h_{t2} and the output y_2.
4. Repeat step 3.
5. Concatenating output sequences $y_1, y_2 ... y_n$ to Y.

Due to the shift register in the convolutional encoder, the adjacent code word is correlated. Because of the process of pass state, we intuitively think the SNND can learn the decoding process equally in each sub GRU model without losing previous code word's information. And the numerical results will be shown in the next chapter.

CHAPTER 5. NUMERICAL RESULTS AND DISCUSSION

In this chapter, we will briefly discuss the numerical results of the SNND in different constellation mapping and explain the reason behind the results. This chapter will be divided into two sections; the first one is to describe what type of system model we used and how to find the best training setting. The second one is numerical results under different constellation mapping and discussion based on the results.

5.1 Parameter Setting

The design of a neural decoder mostly depends on parameter settings, which directly affects neural decoder performance. The neural decoder parameters can be divided into five parts, and each part is indispensable. The first one is the loss function, the second one is the optimizer, the third one is the Metrics, the fourth one is the training epoch, and the last is the training Signal-to-noise ratio (SNR).

For the choice of loss function and optimizer, we referred to Tandler et al. (2019), which provides empirical results. Inspired by Tandler et al. (2019) we chose mean squared error (L2 norm) and Adam optimizer (Benammar and Piantanida (2018)) as our loss function and optimizer respectively. The mean squared error is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{5.1}$$

Gruber et al. (2017) provided a informative metric called normalized validation error(NVE). This metric evaluates neural decoder's performance by normalizing its BER within a certain SNR to the optimal BER obtained by Viterbi decoder with the soft decision. This metric is defined as

$$NVE(\rho) = \frac{1}{S}\sum_{s=1}^{S}\frac{BER_{NND}(\rho,\ \rho SNR, s)}{BER_{Viterbi}(\rho SNR, s)} \tag{5.2}$$

where ρ is SNR of training set, ρSNR denoters the SNR of validation set, and S is the number of SNR points. Inspired by NVE, we straightforward use BER of validation set as our metric, which shown as equation 4.7.

Zhang and Luo (2019) has shown the best training epoch, but during our experiment, we found there could exist an over-fitting problem. The problem can be shown as figure 5.1, which describes the BER performance of the validation set versus the training epoch. It can be easy to find that BER will continue to decrease until epoch 4, and after epoch 4 the curve of BER will continuously rising. Although the curve rise with a turbulence slope, it minimum point is epoch 4. This result clear reveal that we don't need tremendous training epoch, we just need optimal one. The optimal training epoch will be provided shown in table 5.3.

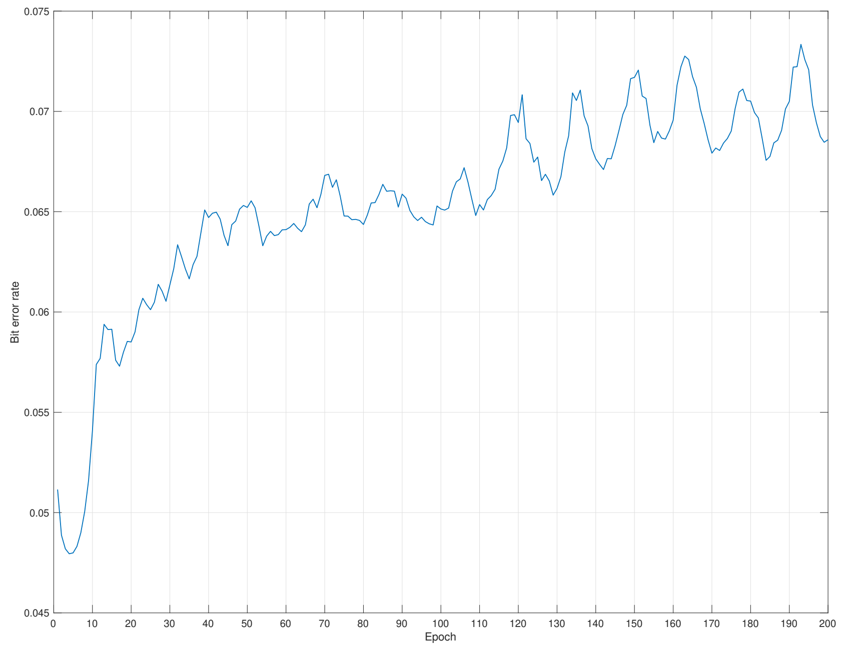


Figure 5.1 epoch vs BER

Gruber et al. (2017) and Kim et al. (2018) has already shown that given a convolutional code and neural decoder, there exists an optimal training SNR. The best training SNR analytical by

Benammar and Piantanida (2018) in later. We intuitively think find the optimal training SNR is a trade-off between training without noise (basically learn the process of encoding) and learning how to deal with noise observation. An example of GRU model training with different SNR results in different performance is shown in figure 5.2 and the training data set used constellation mapping of BPSK. we can observe from figure 5.2 that fixed model training with different SNR will cause varying BER results, and the optimal training SNR under this specific setting is 4dB.

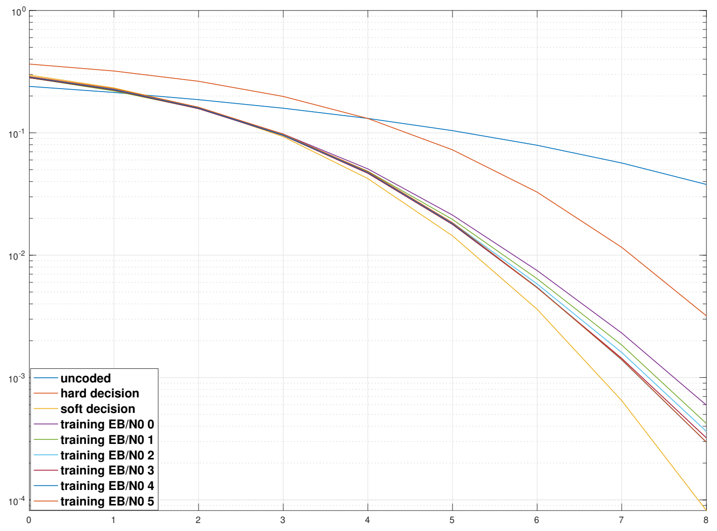


Figure 5.2 BER of different training SNR

In later experiments, we used the MATLAB 2020a to generate data set and used the tensorflow to do our neural network training process. Tensorflow allows to quickly deploy neural network from an abstract point in the python programming language, which hides much of the underlying complexity. Since we support reproducible work, the source code of this thesis will be shown in the appendix.

We chose the most extended block length in IEEE 802.11 specifications as our proposed model's input size is 4096×2, and the input size of the GRU model is 128×2. The model size is shown in table 5.2. The detail of the convolutional code, which we used is shown in table 5.1. The code rate is 1/2 , which corresponds to 2 in the input size of the model. Before testing the optimal training SNR, we need to determine the remaining parameter settings to avoid results inaccuracies. After a lot of experiments, we found the optimal training epoch is only concerned about the training data size, and the the optimal training SNR (measured by E_b/N_0) depends on the range of the testing SNR. Our testing SNR is [0dB-8dB] , the optimal training SNR is 4 dB.The whole parameter settings are shown in table 5.3.

Table 5.1 Characteristics of learned convolutional codes

Type	**Generator Polynomials**	**Code Rate**	**Constraint Length**
NSC	$(o5, o7)_2$	1/2	3

Table 5.2 Model Size

Model	**Input Size**	**Output Size**
GRU model	128×2	128×1
SNND	4096×2	4096×1

Table 5.3 Parameter Settings

Method	**Loss Function**	**Optimizer**	**Metrics**	**Training Epochs**	**Traning** $\mathrm{E_b/N_0}$
GRU model	MSE	Adam	BER	3	4dB
SNND	MSE	Adam	BER	12	4dB

5.2 Numerical Results

In this section, we will show specific parameter settings, and BER performance between the GRU model, the SNND, and Viterbi decoder corresponds to BPSK, QPSK, and 16QAM espectively. These results could demonstrate the SNND we proposed can process long sequence decoding for convolutional code and remain the stable performance with an ideal neural decoder.

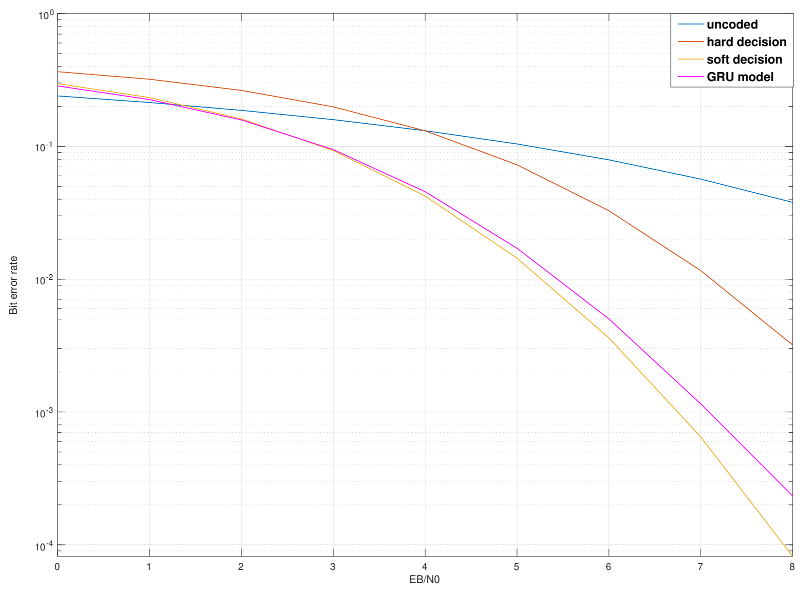


Figure 5.3 Block length 128 BPSK

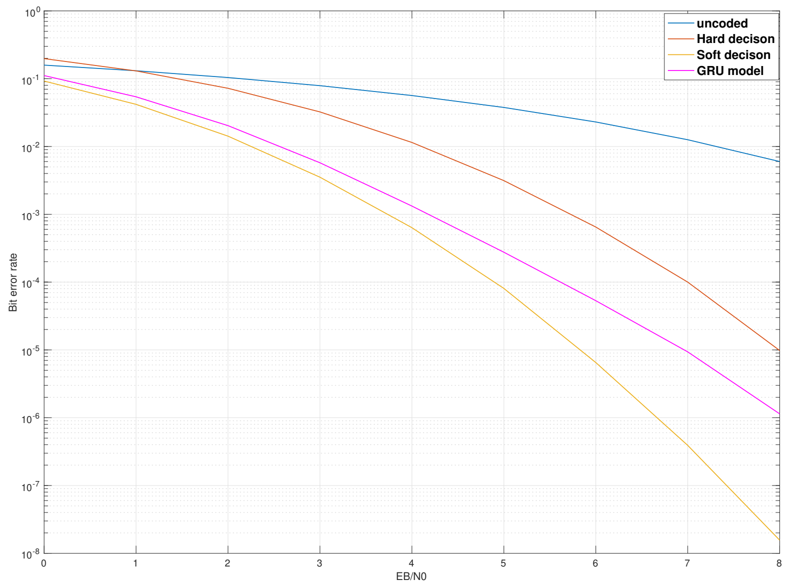


Figure 5.4 Block length 128 QPSK

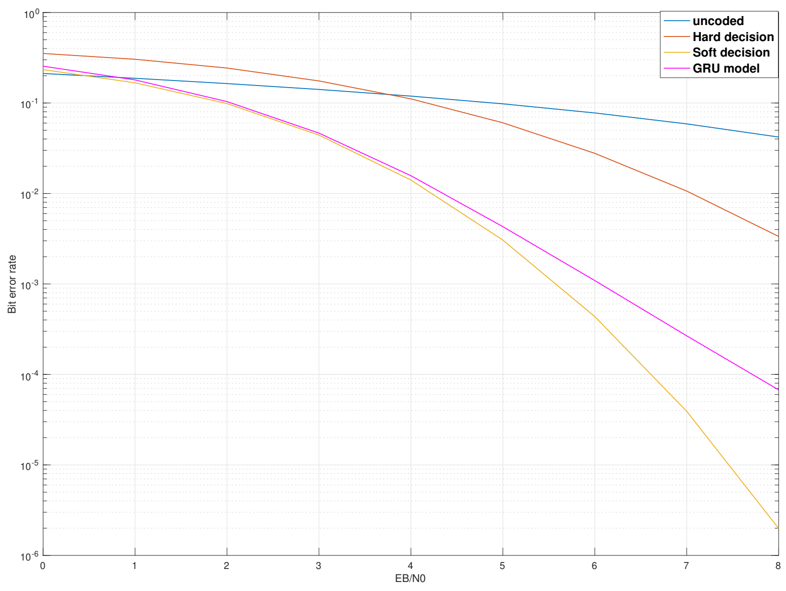


Figure 5.5 Block length 128 16QAM

We first want to show the GRU model's capacity under different constellation mapping because the GRU model is the basis of our proposed model. Our proposed model will not achieve excellent performance if the GRU model performs unsatisfactorily. We use the GRU model to decode the convolutional code with blcok length 128 under constellation mapping of BPSK, QPSK and 16QAM.

The performance of the GRU model under different constellation mapping is shown in figure 5.3, figure 5.4 and figure 5.5 respectively. For an overview, we found the GRU model can achieve a much better performance than the Viterbi hard decision, whatever type of constellation mapping. But the results exists a difference in comparing with the Viterbi soft decision and the GRU model. The GRU model performs almost equal performance with the Viterbi soft decision in BPSK, but the gap with the Viterbi soft decision increases with the modulation order. In 16QAM, the gap between the GRU model and the Viterbi soft decision is much more significant than in BPSK.

Because of the incredible performance of the GRU model, we can start to evaluate our proposed model in dealing with long block length convolutional code.

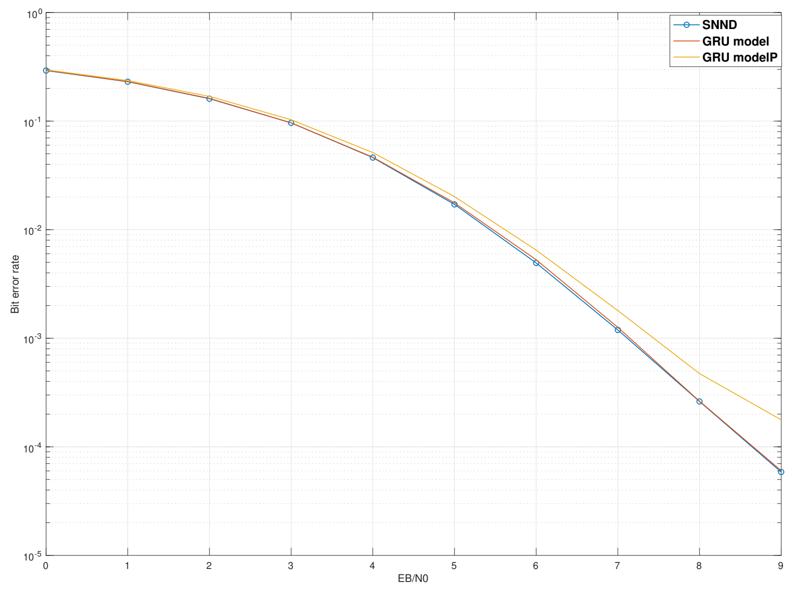


Figure 5.6 Block length 4096 BPSK

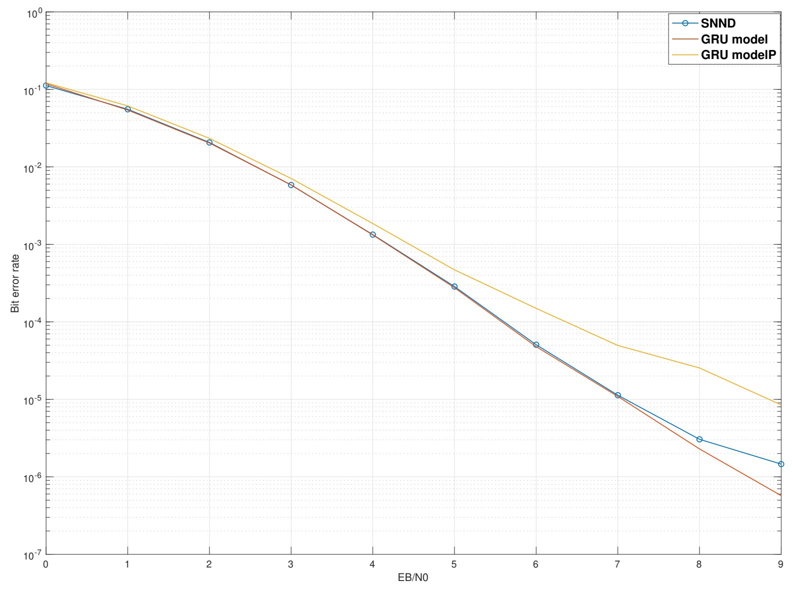


Figure 5.7 Block length 4096 QPSK

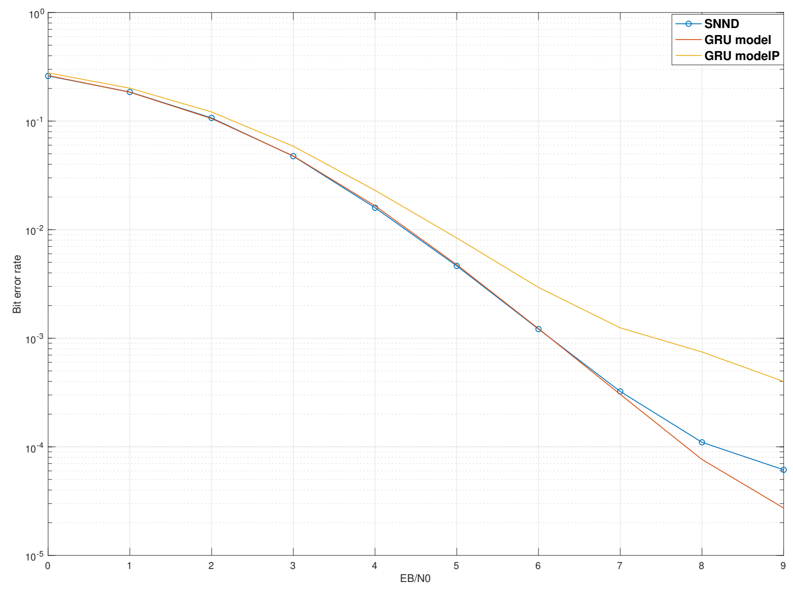


Figure 5.8   Block length 4096 16QAM

In order to distinguish the GRU shown in the above and the partitioned GRU model, we denoted the previous GRU model as the ideal GRU model. The performance of the ideal GRU model, the SNND and the partitioned GRU model with constellation mapping of BPSK, QPSK, 16QAM are shown in figure 5.6, figure 5.7 and figure 5.8 respectively. The decoding block length of the partitioned GRU model and the SNND is 4096. During the experiment, we found the performance of different constellation mapping exhibits difference at 8dB. In order to see more clearly trend, we slightly increase the testing SNR to 9dB. Comparing the performance between the partitioned GRU model and the SNND, the SNND performs better in all constellation mapping. The performance between the ideal GRU model and the SNND need to discuss separately. In BPSK, the ideal GRU model and the SNND showed almost equivalent performance. Along with the modulation order increase, the ideal GRU model performs better than the SNND at high SNR. This trend is the same as the trend of the performance between the ideal GRU model and the Viterbi soft decision.

5.3 Computation Complexity

The computation complexity of Viterbi decoder shown in Lin and Costello (2001). Supposed a (n, k, v) encoder, the viterbi algorithm must perform 2^v add, compare, select (ACS) operations per time unit, one for each state, and each ACS operations involves 2^k additions, one for each branch entering a state. As a result, the computation complexity of the Viterbi algorithm is

$$2^k \cdot 2^v = 2^{k+v} \tag{5.3}$$

assumed the length of information bits is h , the total computation complexity is

$$h \cdot 2^k \cdot 2^v = h \cdot 2^{k+v} \tag{5.4}$$

The mathematical expression of each layers in neural network decoder after training is

$$y = wx + b \tag{5.5}$$

caused our decoder has four layers, first second and third has 256 neurons, respectively. Last layer has 1 neurons. Each neurons has h multiplication and one addition. Hence the total computation complexity of neural network decoder is

$$(3 \cdot 256 + 1)(h + 1) \approx 769h \tag{5.6}$$

5.4 Discussion

From the previous numerical result, we have known the gap between the GRU model and the Viterbi soft decision increases along with the modulation order rise. From the constellation mapping of BPSK to 16QAM, one symbol can map one bit to four bits. As a result, the complexity of the symbol is increased. We used the same model to learn the training data set with each constellation mapping. Due to the fact that the complexity depends on the type of modulation, and the capacity of the model is fixed, the performance of decoding by the GRU model with different constellation mapping will be different. So we think the reason of the gap between GRU model and the Viterbi soft decision is the type of modulation changes. The modulation order increases, the difficulty of learning the decoding process will also increase. As a result, the modulation order increases, the GRU model's gap, and the Viterbi soft decision are bigger. Although the gap will increase, the GRU model still achieves an acceptable result.

Another phenomenon is the performance between the ideal GRU model and the SNND. In the constellation mapping of BPSK, as the number of SNND's sub GRU model increases, the performance of the SNND remains unchanged. And the performance of the SNND is equivalent with the ideal GRU model In the constellation mapping of QPSK and 16QAM, the ideal GRU model performs better than the SNND. In the training process, the SNND will pass the last state of the current sub GRU model to the following GRU model as the initial state. The adjacent convolutional code words are related that will affect each other in the decoding process. The process of passing the last state can keep the integrity of the training noisy code word that enables each sub GRU model to learn the convolutional code completely as the ideal GRU model does. Therefore, each sub GRU model can achieve the performance of the ideal GRU model without loss. So in the constellation mapping of BPSK, the SNND can achieve the performance equal with the ideal GRU model. In the constellation mapping of QPSK and 16QAM, as mentioned above, due to the modulation order increases the training complexity also increases. So that the ideal GRU model's performance can not achieve Viterbi soft decision's performance. Along with the sub GRU model increase, each sub GRU model can not reach the optimal performance so that the errors in each sub GRU model will accumulate to a larger one. As a result, the SNND will perform worse than the ideal GRU model in the constellation mapping of QPSK and 16QAM.

The SNND has reached our expectations that decoding long block length with excellent performance and the performance unchanged along with the sub GRU model increases unlike the SCL model in Cammerer et al. (2017).

CHAPTER 6. SUMMARY AND FUTURE WORK

In this thesis, we proposed a neural network model called the Sequential neural network deocder that can achieve significant performance in decoding long-block-length convolutional codes. In the constellation mapping of BPSK, the performance of the SNND will keep the same as the ideal GRU model. In the constellation mapping of QPSK and 16QAM, the SNND will perform worse than the ideal GRU model. On the whole, the SNND shows an excellent performance in decoding long block length convolutional code, but it still can be improved in three parts, which are the future work of this thesis. The detail of future works are shown as follow.

The first future work is to narrow the gap between the GRU model and the Viterbi soft decision in the modulation of QPSK and 16QAM. The second future work is to mitigate the gap between the SNND and the ideal GRU model in the constellation mapping of QPSK and 16QAM. The third one is to learn the convolutional code with long constraint length. We think one way to address the problems above is to use the method of prior ramp-up, which was mentioned in Tandler et al. (2019).

BIBLIOGRAPHY

Benammar, M. and Piantanida, P. (2018). Optimal training channel statistics for neural-based decoders. *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, page 2157–2161.

Cammerer, S., Gruber, T., Hoydis, J., and ten Brink, S. (2017). Scaling deep learning-based decoding of polar codes via partitioning. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*.

Elias, P. (1955). Coding for noisy channels. *IRE Convention Record*, pages 37–46.

FANO, R. (1963). A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, 9(2):64 – 74.

Forney, G. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268 – 278.

Gruber, T., Cammerer, S., Hoydis, J., and t. Brink, S. (2017). On deep learning based channel decoding. *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, page 1–6.

Hagenauer, J. and Hoeher, P. (1989). A viterbi algorithm with soft-decision outputs and its applications. *1989 IEEE Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond'*.

Hamalainen, A. and Henriksson, J. (1999). A recurrent neural decoder for convolutional codes. *1999 IEEE International Conference on Communications (Cat. No. 99CH36311)*, 2:1305–1309.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hopfield (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79:2554–2558.

IEEE (2016). 802.11-2016 - ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Standard for Information technology*.

Jiang, Y., Kim, H., Asnani, H., Oh, S., Kannan, S., and Viswanath, P. (2020). Feedback turbo autoencoder. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Kim, H., Jiang, Y., Rana, R., Kannan, S., Oh, S., and Viswanath, P. (2018). Communication algorithms via deep learning. *arXiv preprint arXiv:1805.09317*.

Li, H., Shen, Y., and Zhu, Y. (2018). Stock price prediction using attention-based multi-input lstm. *Proceedings of The 10th Asian Conference on Machine Learning*, 95(11):454–469.

Lin, S. and Costello, D. J. (2001). *Error control coding*. Pearson, Prentice hall.

Lyu, W., Zhang, Z., Jiao, C., Qin, K., and Zhang, H. (2018). Performance evaluation of channel decoding with deep neural networks. *2018 IEEE International Conference on Communications (ICC)*.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *he bulletin of mathematical biophysics*, 5(4):115–133.

Omura, J. (1969). On the viterbi decoding algorithm. *IEEE Transactions on Information Theory*, 15(2):177 – 179.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673 – 2681.

Tallini, L. G. and Cull, P. (1974). Neural nets for decoding error-correcting codes. *IEEE Technical Applications Conference and Workshops. Northcon/95. Conference Record*, pages 89–.

Tandler, D., Dörner, S., Cammerer, S., and t. Brink, S. (2019). On recurrent neural networks for sequence-based processing in communications. *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, page 1–7.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260 – 269.

Wang, X.-A. and Wicker, S. B. (1996). An artificial neural net viterbi decoder. *IEEE Transactions on Communications*, 44(2):165–171.

Wozencraft, J. M. (1957). Sequential decoding for reliable communication. *IRE Convention Record*, 5:11–25.

Ye, H., Liang, L., and Li, G. Y. (2019). Circular convolutional auto-encoder for channel coding. *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*.

Zhang, X. and Luo, T. (2019). A rnn decoder for channel decoding under correlated noise. *2019 IEEE/CIC International Conference on Communications Workshops in China (ICCC Workshops)*.

Zhang, Z., Yao, D., Xiong, L., Ai, B., and Guo, S. (2020). A convolutional neural network decoder for convolutional codes. *International Conference on Communications and Networking in China*, pages 113–125.

APPENDIX A. MATLAB CODE

the code for generate data set is shown as follow

```
function[oridata,demodudata] = generate_data(EBN0,M)
k = log2(M);
rate = 1/2;
trellis = poly2trellis(3,[7 5]);%  11,[2473 3217]7,[171 133]5,[35 23]
numSymPerFrame =4096 ;
symbol=1280;
oridata=zeros(symbol,numSymPerFrame);
demodudata=zeros(symbol,numSymPerFrame,2);
for i =1:symbol
    dataIn=randi([0 1],numSymPerFrame,1);
    oridata(i,:) = dataIn;
    dataEnc = convenc(dataIn,trellis);
    txSig = qammod(dataEnc,M,'InputType','bit','UnitAveragePower',true);
    snrdB = EBN0 + 10*log10(k*rate);
    noiseVar = 10.^(-snrdB/10);
    rxSig = awgn(txSig,snrdB,'measured');
    rxDataSoft = qamdemod(rxSig,M,'OutputType','approxllr', ...
        'UnitAveragePower',true,'NoiseVariance',noiseVar);
    rxDataSoft = reshape(rxDataSoft,2,numSymPerFrame).';
    demodudata(i,:,:)=rxDataSoft;
end
end
```

APPENDIX B. PYTHON CODE

the code for training

```
import tensorflow as tf
from tensorflow.keras import Sequential,Model
from tensorflow.keras.layers import ,Dense,GRU, Bidirectional,Concatenate
import scipy.io
import numpy as np
from utlites import biterr
class Decoder(tf.keras.Model):
def __init__(self, rnn_size):
super(Decoder, self).__init__()
self.rnn_size = rnn_size
self.gru1= Bidirectional(GRU(rnn_size,return_sequences=True))
self.gru2 = Bidirectional(GRU(rnn_size, return_sequences=True))
self.gru3 = Bidirectional(GRU(rnn_size, return_sequences=True,return_state=True))
self.dense = Dense(1,activation='sigmoid')
self.concatenate=Concatenate()
def call(self, input):
# out=tf.expand_dims(input,0)
out = self.gru1(input)
out = self.gru2(out)
out, forward,backward = self.gru3(out)
out = self.dense(out)
return out,forward,backward
```

```
class Decoder1(tf.keras.Model):
def __init__(self, rnn_size):
super(Decoder1, self).__init__()
self.rnn_size = rnn_size
self.gru1= Bidirectional(GRU(rnn_size ,return_sequences=True))
self.gru2 = Bidirectional(GRU(rnn_size , return_sequences=True))
self.gru3 = Bidirectional(GRU(rnn_size , return_sequences=True,return_state=True))
self.dense = Dense(1,activation='sigmoid')
self.concatenate=Concatenate()
def call(self, input,forward,backward):
out = self.gru1(input,initial_state=[forward,backward])
out = self.gru2(out)
out, forward,backward = self.gru3(out)
out = self.dense(out)
return out,forward,backward

def loss_func(targets, logits):
mse = tf.keras.losses.MeanSquaredError()
loss = mse(targets, logits)
return loss
optimizer = tf.keras.optimizers.Adam()

mirrored_strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"]
,cross_device_ops=tf.distribute.HierarchicalCopyAllReduce())
with mirrored_strategy.scope():
@tf.function
def train_step(source_seq, target_seq_out):
```

```
loss = 0
with tf.GradientTape() as tape:
n = np.array([i for i in range(128,4224,128 )])
decode,forward,backward = Decoder(source_seq[:,0:128,:])
DECODER=decode
for i in range(len(n)-1) :
n0=n[i]
n1=n[i+1]
decode1, forward, backward = Decoder1(source_seq[:, n0:n1, :], forward, backward)
DECODER=concatenate([DECODER, decode1],axis=1)
loss = loss_func(target_seq_out, DECODER)
variables = Decoder.trainable_variables+ Decoder1.trainable_variables
gradients = tape.gradient(loss, variables)
optimizer.apply_gradients(zip(gradients, variables))
return loss

LSTM_SIZE=256
Decoder = Decoder( LSTM_SIZE)
Decoder1 = Decoder1(LSTM_SIZE)
mat = scipy.io.loadmat('data/BPSK/BPSKtrain128data0.mat')
train_ori=mat['trainori']
train_noise=mat['trainnoise']
mat1 = scipy.io.loadmat('data/BPSK/BPSKtest128data4.mat')
test_ori=mat1['testori']
test_noise=mat1['testnoise']
test_ori = tf.dtypes.cast(test_ori, tf.float32)
test_noise = tf.dtypes.cast(test_noise, tf.float32)
```

```
BATCH_SIZE=128
dataset = tf.data.Dataset.from_tensor_slices((train_ori, train_noise))
dataset = dataset.batch(BATCH_SIZE)
NUM_EPOCHS=200
BER=[]
for e in range(NUM_EPOCHS):
for batch, (train_ori, train_noise) in enumerate(dataset.take(−1)):
train_ori = tf.dtypes.cast(train_ori, tf.float32)
train_noise = tf.dtypes.cast(train_noise, tf.float32)
loss = train_step(train_noise, train_ori)
print('Epoch {} Loss {:.4f}'.format(e + 1, loss.numpy()))
n = np.array([i for i in range(512, 41472, 512)])
decode, forward, backward = Decoder(test_noise[0:512, :, :])
DECODER = decode
for i in range(len(n) − 1):
n0 = n[i]
n1 = n[i + 1]
decode1, forward, backward = Decoder(test_noise[n0:n1, :, :])
DECODER = concatenate([DECODER, decode1], axis=0)
ber = biterr(test_ori, np.round(DECODER))
print('Epoch {} ber {:.4f}'.format(e + 1,ber))
BER =np.append(BER, ber)
postion=np.argmin(BER)
print('epoch',postion+1)

mat = scipy.io.loadmat('data/QPSK/QPSKtest128data9.mat')
test_ori=mat['testori']
```

```
test_noise=mat['testnoise']
test_ori = tf.dtypes.cast(test_ori, tf.float32)
test_noise = tf.dtypes.cast(test_noise, tf.float32)
n = np.array([i for i in range(512, 41472, 512)])
decode, forward, backward = Decoder(test_noise[0:512, :, :])
DECODER = decode
for i in range(len(n) - 1):
n0 = n[i]
n1 = n[i + 1]
decode1, forward, backward = Decoder(test_noise[n0:n1, :, :])
DECODER = concatenate([DECODER, decode1], axis=0)
ber9 = biterr(test_ori, np.round(DECODER))
```