A New Channel Coding Scheme with Machine Learning

Yisu Wang

A project report submitted in partial fulfilment of the requirements
for the degree of Master of Science in Digital Communication Systems

Supervisor: Dr. Alex Gong

Wolfson School of Mechanical, Electrical, and Manufacturing
Loughborough University
Date 29/08/2018

Acknowledgements

I would like to express my gratitude to all those who helped me during the writing of this thesis.

I thank my supervisor Dr. Alex Gong. Without his patient guidance, I couldn't finish this thesis at high quality.

I thank my parents. Without their support, I couldn't begin my academic career.

Finally, I thank my classmates who exchanged from Xidian University with me: Zhang Yuan, Xu Kaiyuan, Chen Hongqiao; and my friend Mao Wenqing. Without their help and encouragement, I couldn't go through this tough period. And I really value their friendships.

Clouds gather, treetops toss and sway; but pour us wine, an old one! That we may turn this dreary day to golden; yes, to golden!

Contents

Abstract

Nowadays, communication engineering field is getting more and more complex and mature, and researchers find it hard to improve the performance by applying expert knowledge. At the same time, deep learning shines in many domains such as computer vision and cyberspace security. An increasing number of researchers have recognized the potential application of deep learning to the physical layer.

We present and discuss a novel coding and modulation scheme with deep learning. It is considering a communication system as an autoencoder. The transmitter and receiver are composed of neural network layers. Then we train those deep neural network layers of the autoencoder and achieve excellent performance for encoding and decoding. In this thesis, we show how to build an end-to-end learning of communication system based on neural networks, and compare the performance of it with the traditional communication schemes (QPSK and BPSK). We also present a discussion of some open research challenges in this area and several kinds of future work needed to be done.

Chapter 1

Introduction

Nowadays, wireless communication systems become increasingly complex. Researchers try to communicate in more complex electromagnetic signal spaces. The amount of information transmitted is getting larger and larger. Thus, research on high-speed modulation and encoding has high application value. In communication engineering domain, researchers design communication systems depending on rich expert knowledge. This engineering field is getting more and more complex and mature, and researchers find it hard to improve the performance by applying expert knowledge. Because of this, an increasing number of researchers have recognized the potential application of deep learning (DL) to the physical layer. Communication systems based on machine learning (ML) or deep learning might provide tangible new benefits.

As a new field of machine learning research, deep learning has received a lot of attention since it had been introduced in 2006. It promotes the development of the field of artificial intelligence. In addition, in this era of Big Data, if researchers demand to extract useful information from massive data, it requires a powerful model to support. At the same time, deep neural networks (DNN) are favoured for its powerful representation capabilities. From this perspective, if a field could be combined with machine learning or deep learning, this area would be developed better and faster. For researchers who are working on communication engineering, it is meaningful to apply deep learning to physical layer for yielding better performance.

Due to the rapid development of deep learning, the application of deep learning has been applied in almost all research fields. In particular, deep learning is widely used in fields such as computer vision (CV) and natural language processing (NLP). One of the characteristics of these areas is that

it is difficult to use rigid mathematical model to accomplish the required practical tasks such as images identification and speech recognition.

In the field of communication engineering, researchers have tried to extend machine learning and deep learning towards communication area in the past, but they rarely focus on physical layer. Machine learning did not cause any fundamental impact on the physical layer. The main reason is that when designing and implementing communication systems, we mainly model the system based on information theory, statistics and signal processing. As long as the system model could fully fit the real-world conditions, we could design an extremely accurate communication system and achieve perfect performance. And then we could implement a powerful algorithm for symbol detection.

Nevertheless, we believe that deep learning could yield significant improvements on the communication engineering. The author in [1] presented a completely new way to design communication system in the physical layer. It is considering a communication system as an autoencoder. We could use deep learning theory to train those deep neural network layers of the autoencoder and achieve excellent performance for encoding and decoding. This is a very useful and insightful way to rethink about communication engineering. In this thesis, we would simulate the autoencoders of [1] and evaluate the performance of those. We also present the representation of the autoencoder with higher dimension, and discuss the problem of representation of the autoencoder. This thesis is based on neural networks and Graphics Processing Unit (GPU) and Keras [12] to simulate communication systems. The autoencoder for communication is a relatively novel and ingenious method. Combining the modulation and encoding of communication signals with the current hot deep learning technology is a relatively innovative idea.

The research topics in this thesis would involve learning transmitter and receiver implementations and further performance evaluation. The methodologies are the algorithm of neural networks, machine learning techniques, GPU parallel computing techniques, and algorithms of autoencoders. These algorithms and simulation techniques are the focus and hotspots of current academic research, and they have high research value in theory and engineering practice. The full text is divided into seven chapters. The main contents of each chapter are briefly described below:

1. The first chapter is an introduction of the full text. It discusses the research background of the topics, and puts forward the work of this thesis, and gives the layout of the full text.

2. The second chapter gives the related work (literature review), sum-

marizes the research results of the predecessors. The related work is mainly about the development of deep learning and the application of deep learning in the field of communication.

3. In the third chapter, for the sake of evaluation, we would take Quadrature Phase Shift Keying (QPSK) for an example, and briefly describe the scheme of this technique. This is not the focus of our attention, because the aim of introducing QPSK is to compare the performance of autoencoders and traditional communication systems.

4. The fourth chapter gives the theoretical background of this thesis, which is the theory of neural works. This chapter briefly introduces several basic concepts of neural networks, activation functions, training algorithm, and the relationship with deep learning.

5. The fifth chapter describes the application scenarios of neural networks in this paper, which are the autoencoders for communication system. The transmitter and receivers of the system are represented by one deep neural network. This neural network could be trained as an autoencoder, because the process of training is unsupervised.

6. The sixth chapter presents the environment setup. It includes the introduction of Python and Matlab and instructions of how to build an antoencoder by Keras.

7. The seventh chapter presents the simulation results of autoencoders using different parameters. And we describe how to build the neural networks in details, then we discuss the difficulties of implementing and analysing these autoencoders. In addition, we also present several kinds of future work needed to be done.

8. The eighth chapter is a summary and outlook of the full text. It outlines the main work and innovations of the full text, and points the direction for the future research.

Chapter 2

Literature Review

Firstly, we would introduce the history of development about deep learning, and then introduce the application of DL, especially in the wireless communication engineering.

2.1 The Development of Deep Learning

In 1989, Cybenko pointed out that any bounded continuous function could be approximated with arbitrary precision by a neural network with two layers in [2]. One output layer uses linear activation function, and one hidden layer uses sigmoid activation function. But the neural network with only two layers has limitations such as poor generalization capabilities and lack of accuracy, then researchers wish to reduce the number of neurons in the network layers to make the models achieve better generalization capabilities. Specifically, to reduce the number of neurons in the network, many models could be used to implement deep architectures. The most widely used of deep architectures is the deep neural network (DNN).

Prior to 2006, the traditional method of training neural networks with multiple layers was:

1. To randomly initialize the weight of the deep neural network.
2. To use the back propagation algorithm to optimize the objective function by learning from the training dataset [29].

Through experiments it has been found that deep neural networks usually have poor generalization ability, even worse than shallow neural networks [3]. The authors in [4] give three reasons why the deep neural networks trained by the traditional algorithm could not achieve the expected effect:

1. There is a lack of labelled training data. In real-world applications, labelled data is often scarce. Therefore, although deep learning has strong power on fit any bounded continuous function, using a supervised method to train such models on training dataset with insufficient labelled data would lead to overfitting.

2. Sometimes there exists a problem of convergence to a sub-optimal value. When we use traditional methods to train shallow neural networks, it is usually possible to converge the parameters of the neural networks to a reasonable range. However, when we use this method to train deep neural networks, it is difficult to avoid the problem of convergence to a sub-optimal value. When we start with a random initialization point and use SGD algorithm to search, we usually fall into a bad local extremum.

3. There exists a vanishing gradient problem. When we use back propagation algorithm to calculate the partial derivative of the objective function, as the network depth increases, the gradient value of the back propagation would be drastically reduced. Thus, the derivative of the objective function would cause little influence to the weights of the initial layers. And the weights of the first few layers would change very slowly, and the model could not learn effectively from the training dataset.

The training problem of deep neural networks was not solved until 2006 by the greedy layer-wise training [5]. In [5], the algorithm successfully trains a deep autoencoder, and obtains a good effect of reducing the data dimensionality. At the same time, this algorithm is successfully implemented in multiple data sets, and obtained the best classification effect at that time. Then, in the same year, Hinton et al. successfully implement a deep belief networks (DBN) [6] by applying this training algorithm. So far, the process of training deep neural networks using greedy layer-wise training algorithm is called deep learning.

2.2 The Application of Deep Learning

Since very recently, open-source DL software libraries (e.g., Caffe [7], MXNet [8], TensorFlow [9], Theano [10], Torch [11], Keras [12]), and powerful specialized hardware, such as field programmable gate arrays (FPGAs) and Graphics Processing Unit are cheaply and readily available. Thanks to

these rapid developments, the applications of deep learning are applied to almost every research domain [13-17]. Especially, deep learning shines in domains such as computer vision and natural language processing, which are difficult to characterize practical tasks with rigid mathematical models.

In communications, researchers have tried to extend machine learning and deep learning towards communications in the past, but they mainly focus on cyberspace security [18-20].

Although several researchers have also addressed problems related to the physical layer with machine learning such as channel modelling and prediction, equalization, quantization [21-22], etc., machine learning did not cause any fundamental impact on the physical layer. The main reason is that the way we design and implement communications systems is generally depended on the complex and mature expert knowledge. Based on information theory, statistics, and signal processing, as long as the system model sufficiently characterizes real effects, we could design extremely accurate communication systems that enable robust algorithms for symbol detection.

However, the authors in [1] presented a completely new way to think about communications systems design by representing a communication system as an autoencoder, which is a deep neural network (NN) typically used to learn how to reconstruct the input at the output. In order to incorporate expert knowledge in the deep learning, [1] also introduces the concept of radio transmitter networks (RTN), a different radio receiver model to improve the performance of autoencoder. Finally, [1] illustrates that deep learning could be useful tools applied to improve current wireless communications. And when channel models are difficult to derive, researchers could turn to deep learning or other machine learning techniques from traditional signal processing algorithms to deduce the channel.

Based on these ideas from [1], the authors in [23] implemented a communication system using only deep neural networks by software-define-radios (SDRs). The results from [23] demonstrate that the autoencoder idea could be implemented in the reality. To implement this fascinating novel antoencoder concept using SDRs, the authors in [23] extended the existing concepts toward continuous signal transmission, which entails the receiver synchronization issue. And finally, they overcame this problem by introducing another neural network layer for frame synchronization.

Some other examples of deep learning tools applied to address problems in physical layer include detection of data sequences [24], modulation recognition [26], compressed sensing [26], [27], learning of encryption/decryption schemes for an eavesdropper channel [28]. The authors in [38] also presented a literature review about applying deep learning into wireless communication

engineering. Nowadays, there are two main different viewpoints of applying deep learning to the communication systems in these papers. The goal is to either completely replace existing communication algorithms with deep learning, or to apply deep learning only for improving them.

Chapter 3

Traditional Communication System

Because this research mainly focuses on applying deep learning to communication systems, the goal of this paper is to show electronic engineers how to build a communication system only by neural networks, which is a novel concept. Since most electronic engineers already know traditional schemes of modulation and encoding, the writer of this paper wouldn't write too much about building a traditional communication system. To make readers who don't have the knowledge about traditional communication system schemes, this section would briefly introduce the basics of quadrature phase shift keying (QPSK), which is a common form of phase modulation.

For all forms of MPSK, all information of the messages would be encoded in the phase of the signal at the transmitter side. Thus, the transmitted signal (the MPSK signal) has an one-to-one correspondence between M-ary symbols and the M signals with different carrier phases. In addition, MPSK could be represented by constellation points in two-dimensional space.

In QPSK system, the transmitter uses four different phases of the carrier to characterize the information. QPSK is a phase modulation technique at M=4. The phases of QPSK transmitted signal are respectively $45^o, 135^o, 225^o, 275^o$. The constellation diagram of QPSK is shown in Figure. 3.1. The input data of the modulator is a binary digital sequence. In order to match these four different phases of carriers, we need to transform the binary symbol into quaternary symbol. It means that we need to divide every two bits of the binary digital sequence into a group. There are four combinations: 00, 01, 10, 11. Each quaternary symbol is composed of two binary information bits. Each modulation in QPSK could transmit two information bits. The demodulator determines the transmitted information

bits by the constellation diagram and the received phase of the carrier signal.

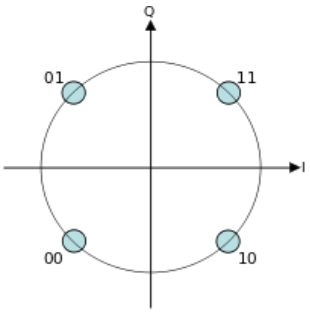

Figure 3.1: The constellation diagram of QPSK

For QPSK demodulator, a very popular scheme is coherent demodulator [34]. But this research mainly focuses on analysing the performance of QPSK, it would not explain every detail about demodulation of QPSK. If any readers are interested in every detail of implementing QPSK, there is a material that could provide a detailed reference on QPSK [34].

In order to provide a comparison with autoencoder, we simulated QPSK system and analysed the performance of it using MATLAB. Suppose the two-sided power spectral density of the additive white Gaussian noise is $N_0/2$, the average energy per symbol of the transmitted signal is E_s. The source code could be found in Appendix.

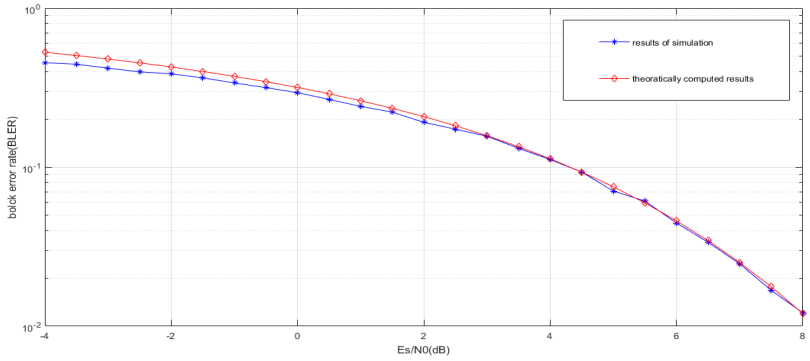

Figure 3.2: The performance of QPSK

We computed the value of block error rate (BLER) with different value of E_s/N_0 in Figure. 3.2. There are two lines in Figure. 3.2. The blue line

is the result of QPSK performance computing by Monte Carlo simulation. The red line is the results of QPSK performance by theoretical calculation.

Chapter 4

Deep Learning Basics

Deep learning is a branch of machine learning. Some of the most successful deep learning methods include deep neural networks (NN), convolutional neural networks (CNN), deep belief networks (DBN), and recurrent neural networks (RNN). In recent years, deep learning has become extremely popular and has been successfully applied in the fields of computer vision, image processing and recognition, speech recognition and natural language processing. In these fields, deep learning completely surpasses traditional machine learning methods such as support vector machine (SVM), Hidden Markov Model (HMM).

In 2006, Geoffery Hinton of the University of Toronto in Canada published a paper [5] in Science and related journals. And it's the first time for the writer of [5] to propose the concept of deep belief network. The deep belief network applies the pre-training process to find the weights of the neural networks. The weights found by pre-training process are close to the optimal solution value, and then the entire time for training neural networks with multiple layers is also reduced through fine-tuning technology. Neural networks use neurons to tag or cluster raw data. If we wish to feed neural networks the real-world data such as images, speech, and text, we must transform those data into what the neural network could recognize. For data that is not labelled, the neural network could find similarities from the input vectors to cluster the data. If the data is labelled, the neural network could be a classifier and usually achieve excited performance. Furthermore, the neural network could extract data features by applying autoencoder and then classify them with other machine learning algorithms. In this way, the neural network could be regarded as a part of machine learning application system. A network based on deep learning is a particular neural network

Table 4.1: List of Layer Types

Name	$f_l(\boldsymbol{r_{l-1}};\boldsymbol{\theta}_l)$	θ_l
Dense	$\sigma(\boldsymbol{W}_l\boldsymbol{r}_{l-1}+\boldsymbol{b}_l)$	$\boldsymbol{W}_l,\boldsymbol{b}_l$
Noise	$\boldsymbol{r}_{l-1}+\boldsymbol{n}$	none
Dropout	$\boldsymbol{d}\odot\boldsymbol{r}_{l-1}$, $d_i\sim$Bern(α)	none
Normalization	e.g., $\frac{\sqrt{N_{l-1}}\boldsymbol{r}_{l-1}}{\|\boldsymbol{r}_{l-1}\|_2}$	none

composed of multiple layers. Each neural network layer consists multiple neurons (In deep learning theory, neuron could be also called node). The neuron is the basic operator in the neural network. It combines the values of input vector and the weights, and computes the sum of the product of these values. The resulting value is the input of the activation function of the neuron. And finally the result of the activation function calculation would cause the influence of the signal in the neural network.

A feedforward Neural Network with L layers is to describe a mapping $f(\boldsymbol{r_0};\boldsymbol{\theta}) : \mathbb{R}^{N_0} \mapsto \mathbb{R}^{N_L}$, which is an input vector $\boldsymbol{r_0} \in \mathbb{R}^{N_0}$ to an output vector $\boldsymbol{r_L} \in \mathbb{R}^{N_L}$, and there are through L iterative processing steps:

$$r_l = f(\boldsymbol{r_{l-1}};\boldsymbol{\theta}_l),\ l = 1,...,L \tag{4.1}$$

where $f(\boldsymbol{r_{l-1}};\boldsymbol{\theta}_l) : \mathbb{R}^{N_{l-1}} \mapsto \mathbb{R}^{N_l}$ is the mapping carried out by the lth layer. This mapping depends on the output vector r_{l-1} and a set of parameters $\boldsymbol{\theta_l}$. This work presents that $f(\boldsymbol{r_{l-1}};\boldsymbol{\theta}_l)$ has the form

$$f(\boldsymbol{r_{l-1}};\boldsymbol{\theta}_l) = \sigma(\boldsymbol{W_l}\boldsymbol{r}_{l-1}+\boldsymbol{b}_l) \tag{4.2}$$

the lth layer is called *dense* or *full-connected* layer, where $\boldsymbol{W}_l \in \mathbb{R}^{N_l} \times \mathbb{R}^{N_{l-1}}, \boldsymbol{b}_l \in \mathbb{R}^{N_l}$, and $\sigma(\cdot)$ is an *activation* function, and $\boldsymbol{\theta}_l = \{\boldsymbol{W}_l, \boldsymbol{b}_l\}$. Common activation functions and layer types are listed in Table. 4.1 and Table. 4.2 respectively.

This thesis uses labelled training data to train neural networks. For instance, the labelled data is a set of input and output vector pairs $(\boldsymbol{r}_{0,i}, \boldsymbol{r}^*_{L,i}), i = 1,...,S$, where $\boldsymbol{r}^*_{L,i}$ is the desired output vector and $\boldsymbol{r}_{0,i}$ is the input vector.

The training process is mainly to reduce the loss to minimum value:

$$\boldsymbol{L}(\boldsymbol{\theta}) = \frac{1}{S}\sum_{i=1}^{S} l(\boldsymbol{r}^*_{L,i}, \boldsymbol{r}_{L,i}) \tag{4.3}$$

Table 4.2: List of Activation Functions

Name	$[\sigma(\boldsymbol{u})]_{\mathrm{i}}$	Range
linear	u_i	$(-\infty, \infty)$
ReLU	$\max(0, u_i)$	$[0, \infty)$
tanh	$\tanh(u_i)$	$(-1, 1)$
sigmoid	$\frac{1}{1+e^{-x}}$	$(0, 1)$
softmax	$\frac{e^{u_i}}{\sum_j e^{u_j}}$	$(0, 1)$

Table 4.3: List of Loss Functions

Name	$l(\boldsymbol{u}, \boldsymbol{v})$
MSE	$\|\boldsymbol{u} - \boldsymbol{v}\|_2^2$
Categorical cross-entropy	$-\sum_j u_j \log(v_j)$

where $l(\boldsymbol{u}, \boldsymbol{v}) : \mathbb{R}^{N_L} \times \mathbb{R}^{N_L} \mapsto \mathbb{R}$ is the loss function; and $\boldsymbol{r}^*_{L,i}$ is the output and we use $\boldsymbol{r}_{0,i}$ as input. Several commonly used loss functions are presented in Table. 4.3.

Then we use the most popular stochastic gradient decent (SGD) algorithm to find sets of parameters θ. The SGD starts with $\theta = \theta_0$ where θ_0 is some random initial parameters and then updates θ iteratively as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla \tilde{L}(\boldsymbol{\theta}_t) \tag{4.4}$$

where the learning rate $\eta > 0$, and $\tilde{L}(\theta_t)$ is the approximation of the loss function.

Note that detailed description of Neural Networks is presented in [29]. This paper defines and trains neural networks by the existing deep learning libraries presented in Chapter 2 (Literature Review). To simulate autoencoder concept from [1], this work mainly uses TesnsorFlow [9] and Keras [12].

Once the deep learning algorithm was proposed, it gained a lot of attention and quickly achieved great success in the fields of computer vision, and natural language processing. However, due to the complexity of deep learning, no theoretical mechanism behind this success has yet been explained. Some researchers try to answer why unsupervised pre-training could help neural networks training, and then use experiments to confirm the validity of their conjectures. The experimental results of [35-36] show that for a

neural network with only one single hidden layer, if we use unsupervised pre-training, it could achieve lower training error values than random-initialized training. This conclusion indicates that unsupervised pre-training could be considered as a better optimizer. But for neural networks with deep hidden layers, this conclusion is not appropriate. In this research, we wouldn't apply pre-training and fine-tuning technology. The traditional training method is enough to train the autoencoders for end-to-end communication systems.

In neural networks, each layer performs feature recognition based on the output vector of the previous layer. The complexity and abstraction of features are directly related to the number of neural network layers. The more the number of the neural network layers, the more the complexity of the features of node increases. It makes neural networks have great advantages in processing large-scale and high-dimensional input data. Most importantly, through autoencoder, neural networks are able to discover potential features in unlabelled, unstructured data. The autoencoder would be introduced in Chapter 5 (End-To-End Learning of Communication Systems).

Chapter 5

End-To-End Learning of Communication System

In this section, we simulate a communication system using only deep neural networks. The main idea is to design a communication system by interpreting it as an autoencoder trained by SGD. This section would firstly reproduce some results of [1], then explain those results carefully and present several novel simulations about that idea. Autoencoder is an unsupervised feature detection model. It could learn the feature of the input data. The model belongs to the artificial neural network (ANN) and applies back propagation algorithm to optimize the results. Specifically, the autoencoder attempts to learn an identity function that the output value of the model is equal to or as close as possible to the input value of the model.

In a communications system, the simplest form consists three parts: a transmitter, a receiver, and a channel. The messages sent by the transmitter are reconstructed by the receiver over a physical channel. The whole process of communication which reconstructs the messages is shown in Figure. 5.1. This figure is from [38].

The transmitted message $\boldsymbol{s} \in \mathcal{M} = 1, 2, ..., M$ is one out of M possible messages, and the transmitter makes n discrete uses of the channel. At the transmitter side, it applies the transformation $f : \mathcal{M} \mapsto \mathbb{R}^n$. The transformation f maps the message $\boldsymbol{s}$ to the transmitted signal $\boldsymbol{x} = f(\boldsymbol{s}) \in \mathbb{R}^n$. Generally, the transmitted signal $\boldsymbol{x}$ should be robust enough to be against the channel with noise. And $\boldsymbol{x}$ should be imposed certain constraints due to the transmitter's hardware. Usually, there are three common constraints: an energy constraint that demands $\|\boldsymbol{x}\|_2^2 \leq n$, an amplitude constraint demands $|x_i| \leq 1, \forall i$, and an average power constraint demands $\mathbb{E}[|x_i|] \leq 1, \forall i$.

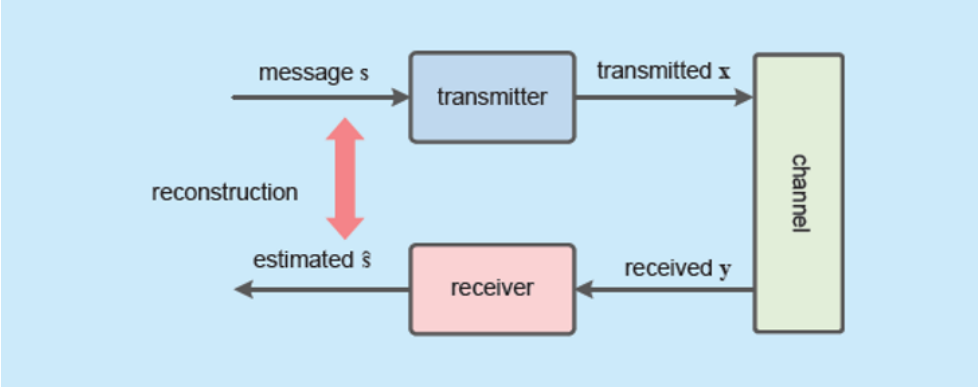

Figure 5.1: A simple form of communication system that reconstructs the the transmitted message at the receiver side

We apply the notation of [1]: the communication rate of the system is $R = k/n$ (bit/channel use), $k = \log_2(M)$. And (n, k) means that the transmitter sends one out of 2^k messages via n channel uses. In addition, $\boldsymbol{y}$ is the received n-demensional signal noised by a channel. To describe the channel, we use a conditional probability density function $p(\boldsymbol{y}|\boldsymbol{x})$ to represent the channel. At the receiver side, the receiver applies the transformation $g : \mathbb{R}^n \mapsto \mathcal{M}$ to reconstruct the transmitted message $\boldsymbol{s}$ as $\tilde{\boldsymbol{s}}$.

Therefore, we could consider the communication as a process of end-to-end reconstruction problem. The transmitter sends messages and the receiver reconstructs the messages over a physical channel. Based on DL theory, this reconstruction process could be considered as a particular type of autoencoder. This autoencoder could achieve global optimization for the transmitter and receiver over a physical channel. In this way, we apply fully connected dense layers to represent the transmitter and the receiver. To represent the AWGN channel between the transmitter and the receiver, we would apply a noise layer with a certain variance.

Usually, if we add some constraints of the networks of the autoencoder, the network could find some interesting structures from the input vectors. For example, we could make the number of neurons of hidden layer smaller than the dimension of the input vector. It is equivanlent to letting the autoencoder learn a compressed representation of the input vectors. In this way, the original data could be reconstructed with less error through the compressed representation. If each feature of the input vectors is a completely independent variable, then the compressed representation will be very difficult to learn. If there is a certain correlation between the features

of the input vectors, then the autoencoder could discover these correlations in the input vectors. At this time, if the output vectors of the hidden layer are added to the sparsity constraint [37], the autoencoder could also learn some interesting structures from the input vectors.

However, in this case, the purpose of applying the concept of autoencoder for communication system is to jointly optimize the transmitter and the receiver over a physical channel. At the transmitter side, the autoencoder seeks to learn how to represents the messages $\boldsymbol{s}$ by applying the transformation $f : \mathcal{M} \mapsto \mathbb{R}^n$. The representations $\boldsymbol{x}$ should be robust with respect to the impairments of the physical channel. In this way, the transmitted messages could be reconstructed with small probability error due to the noise, fading, and distortion of the channel. At the receiver side, the optimization of estimation of the original messages can be done by training neural networks using SGD.

There is an example of autoencoder for an end-to-end communication system in Figure. 5.2. The figure is from [38]. The transmitter firstly uses an M-dimensional one-hot vector to represent $\boldsymbol{s}$. Then the M-dimensional one-hot vector $\boldsymbol{s}$ would be fed into multiple dense layers followed by a normalization layer. The normalization layer ensures that physical constraints on the generated N-dimensional vector $\boldsymbol{x}$ are met. We use an additive noise layer to represent the physical channel. The variance β of the noise layer is a fixed value $2RE_b/N_0$. E_b denotes the energy per bit and N_0 denotes the noise power spectral density. We also apply several dense layers to the receiver. As same as the most part of deep learning for classification, at the receiver side, its last layer applies a softmax activation to generate the decoded messages $\tilde{\boldsymbol{s}}$. The output $p \in (0, 1)^M$ of the dense layer with softmax activation is an M-dimensional probability vector. The sum of all elements of $\boldsymbol{p}$ is equal to 1. Then the value of the element of $\boldsymbol{p}$ with highest probability corresponds to the estimation $\tilde{\boldsymbol{s}}$. The autoencoder-based communication system could learn a joint optimization of coding and modulation scheme.

With the explanations of the autoencoder, we present the corresponding pseudo-code in Algortithm 1.

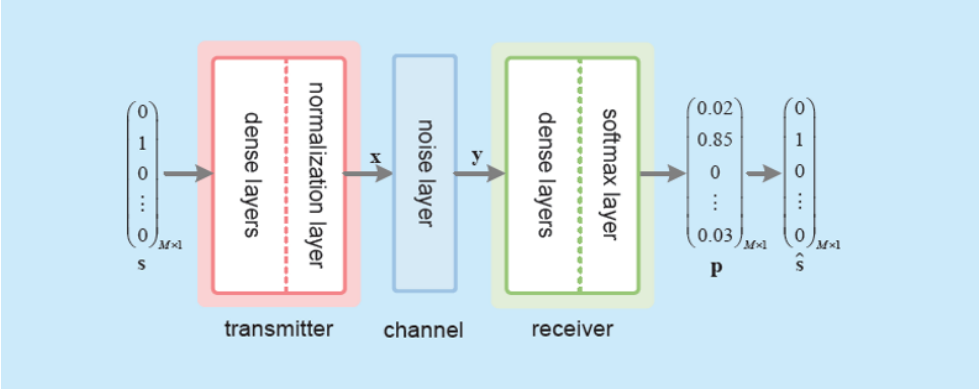

Figure 5.2: A simple autoencoder for an end-to-end communication system.

Table 5.1: Paramaters of Autoencoder

Layer	Output Dimensions
Input	M
Multiple dense layers + ReLU	M
Dense layer + linear	M
Normalization	n
Noise	n
Multiple dense layers + ReLU	M
Dense layer + softmax	M

Algorithm 1 Build and Train an Autoencoder.

1: Generate Batch of one-hot vectors:

$$data = \boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_N \in \mathbb{R}^{n \times N} \tag{5.1}$$

2: Define autoencoder and its layer. The parameters are listed in Table 5.1.

3: Train the autoencoder by feeding it with $data$.
Features = $data$; Labels = $data$.
/*Note: The variance E_b/N_0 is a fixed value: 7 dB.
Training is done using Adam [32] with learning rate 0.001.*/

4: Plot learned constellations.
/*Note: If the constellation is for higher dimension like (7,4) autoencoder, we should use a two-dimensional t-distributed stochastic neighbour embedding (t-SNE) [33] to depict the higher dimensional signal representations. */

5: Calculate the block error rate (BLER) versus different E_b/N_0 value. And plot the BLER curve.

Chapter 6

Enviroment Setup

The programming languages of this research are Python and MATLAB. Because MATLAB toolboxes about communication engineering are professionally developed, rigorously tested, it is extremely easy produce the results of the simulation of traditional modulation and encoding methods.

To implement autoencoders for communication systems, we build and train large neural networks by Python. This paper applies several machine learning libraries such as TensorFlow, Keras, which allow for high level algorithm definition in Python. With just several lines of Python code using Keras, we could build machine learning models without having to be an expert.

Keras is a deep learning framework based on Python implementation, which is an open source. Researchers could use Keras to develop or deploy applications. The core algorithm of these applications is based on deep learning. Keras is suitable to deal with massive data of the Internet, including multimedia data such as voice, image and video. If we use GPU to operate Keras programs, the speed of computing usually would increase.

Keras provides a number of popular models. By fine-tuning these models, researchers don't have to rewrite a large amount of programs during developing new applications. The applications would be developed quickly and efficiently. This is also the trend of software application development. Given by these advantages of Keras, it has become one of the most widely used deep learning frameworks for researchers and developers.

The operation system was Window7. And the environment was done in Anaconda. Anaconda is an environment manager, a Python distribution. We use Spyder to compile the programs. Spyder is one package in Anaconda, which is an integrated development environment (IDE) for Python.

It is a powerful numerical computing environment with advanced editing, interactive debugging. Spyder includes lots of popular Python libraries such as Numpy, Scipy, and matplotlib. The user interface (UI) of Spyder is a little similar to MATLAB.

To install Anaconda, go to the official website [39] to download $Anaconda3-5.2.0-Windows-x86\_64.exe$ and install Anaconda Python with the prompts of this executable file.

To install Keras and TensorFlow with conda run, we need to open Anaconda Prompt and input:

```
conda install -c conda-forge tensorflow
conda install -c conda-forge keras
```

The models in Keras could be mainly divided into two types: the Sequential model, and the Model class used with the functional API (application programming interface). The Sequential is easy to learn, and it is a linear stack of layers. To build neural networks, we could simply add layers through *.add*() method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

To add layers into neural networks in a Sequential model, we need to state the parameters of the layers clearly and appropriately. The parameter types include the dimension of input vector, the dimension of output vector, the activation function and so on. It is enough to simulate autoencoders for end-to-end communication system by using the Sequential model. But we would apply the Model (functional API) to simulate autoencoders. Because autoencoders for communication system with multiple users couldn't be implemented by the Sequential model, it's easier to extend end-to-end system to systems with multiple transmitters and receivers by applying Model (functional API).

The Keras functional Model is a method to build complex models such as models with multiple output vectors and models with shared layers. As long as the structure of the model is complex, or if the model requires more than one output vector, the developers should always choose a functional Model. The Model (functional API) is the most extensive type of deep learning model. The Sequential model is just a particular case of the functional Model.

There is a simple example of building a Sequential model through the way of functional Model:

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes # the Input layer and three
    Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels) # starts training
```

Chapter 7

Results and Discussion

7.1 Results and Analysis

We propose the simulation results of four autoencoders using different parameters (n, k).

To build the neural networks of autoencoders, we use the functions of Keras to generate the dense layers, noise layers, and normalization layers. The code of building neural networks of autoencoder (n, k) is listed as the following:

```
input_signal = Input(shape=(M,))
encoded = Dense(M, activation='relu')(input_signal)
encoded1 = Dense(n_channel, activation='linear')(encoded)
encoded2 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.l2_normalize(
    x, dim=1))(encoded1)
EbNo_train = 5.01187 #  coverted 7 db of EbNo
encoded3 = GaussianNoise(np.sqrt(1/(2*R*EbNo_train)))(encoded2)
decoded = Dense(M, activation='relu')(encoded3)
decoded1 = Dense(M, activation='softmax')(decoded)
autoencoder = Model(input_signal, decoded1)
adam = Adam(lr=0.01)
autoencoder.compile(optimizer=adam, loss='categorical_crossentropy
    ')
```

After building the autoencoder, we could get the summary about the autoencoder. And we during traning those DL-based communication systems, we could acquire the iterations' situation. These information could be found in [40]. And all source code and simulation results of this thesis could be found in [40].

Figure. 7.1∼7.3 show the constellations produced by autoencoders for different parameters (4,7). The constellation points correspond to the learned representations x of the input vectors.

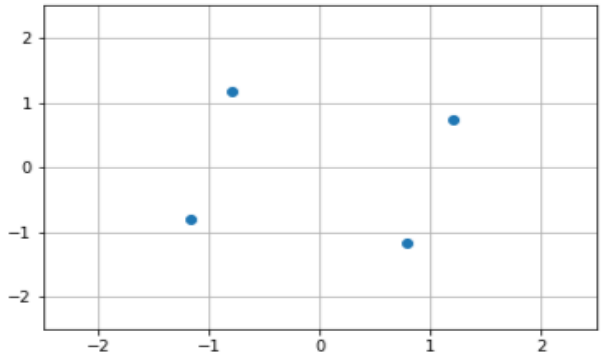

Figure 7.1: Constellations of Autoencoder $(2, 2)$

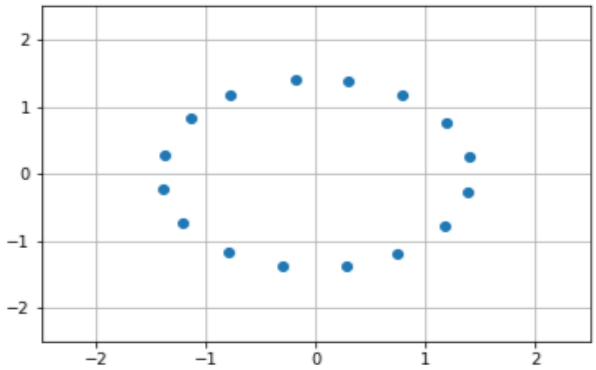

Figure 7.2: Constellations of Autoencoder $(2, 4)$

In Figure. 7.1 for $(2, 2)$, the constellation diagram is similar to the classical QPSK constellation except the constellation points are rotated. Later we would compare the performance about autoencoder $(2, 2)$ and QPSK. To evaluate the performance of these two communication systems, we measure the values of BLER at different SNR.

Similarly, Figure. 7.2 shows the constellation points of system of autoencoder with parameters $(2, 4)$. In [1], the author made a mistake. It should be $(2, 4)$ system which leads to 16-PSK constellation with a little rotation, but the author wrote $(4, 2)$ system. The constellation points of a

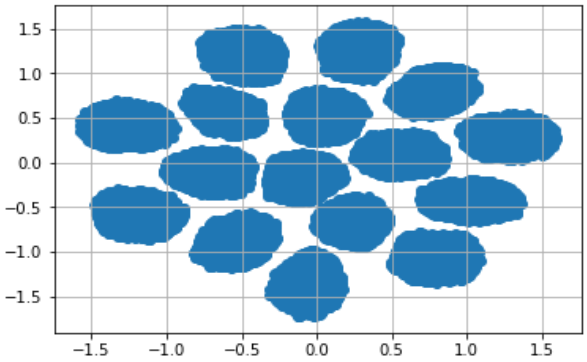

Figure 7.3: Constellations of Autoencoder $(7, 4)$

$(4, 2)$ system couldn't be drawn directly by 2-dimensional diagram, because the learned representations $\boldsymbol{x}$ is a four-dimensional vector.

In Figure. 7.3, to depict the seven-dimensional representations, we apply a two-dimensional t-SNE of the observations $\boldsymbol{y}$ instead. The vector $\boldsymbol{y}$ is the representation $\boldsymbol{x}$ plus the white Gaussian noise. Figure. 7.3 shows that the t-SNE of $\boldsymbol{y}$ leads to an arrangement of sixteen clusters. The running time to compute t-SNE of seven-dimensional messages is quite long. We take almost 48 hours to acquire the results of Figure. 7.3. Thus, depicting representations becomes increasingly impractical and time-consuming for higher dimension. It would cause difficulty for researchers to analyse the distribution of the representations. To circumvent this problem, we tried to use GPU to run our programs. However, the running time for training and testing became shorter, but the running time for computing t-SNE was the same. The reason is that t-SNE algorithm is a non-linear technique for dimensionality reduction. Another possible way to solve this problem is to use other dimensionality reduction algorithms such as principal component analysis (PCA), myelodysplastic syndromes (MDS), etc. But we spend too much time on reproducing the author's results, we don't have enough time to test the performance of other dimensionality reduction algorithms. We will do this work in the future research.

Figure. 7.4∼7.6 present the block error rate (BLER) of autoencoders for different (n, k). The results show that the autoencoders could learn how to encode and decode without any prior knowledge. A single layer for encoder part is enough to learn how to map from messages to transmit vectors, but we apply two layers to help the encoder reduce the likelihood of convergence to sub-optimal minima. In this way, it prevents the solutions to emerge as

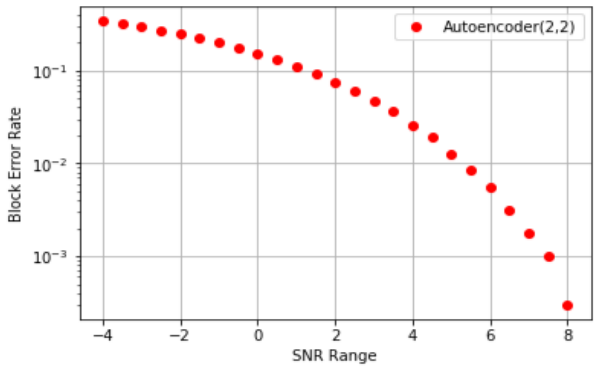

Figure 7.4: BLER versus E_b/N_0 for Autoencoder $(2, 2)$

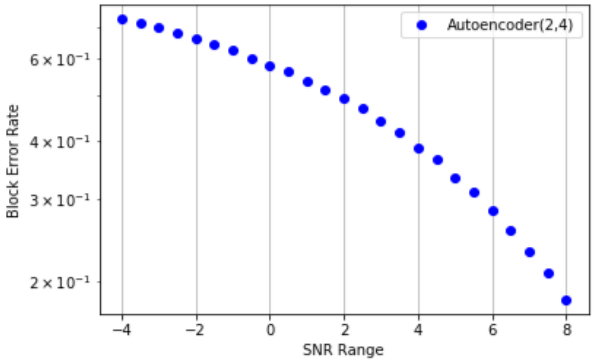

Figure 7.5: BLER versus E_b/N_0 for Autoencoder $(2, 4)$

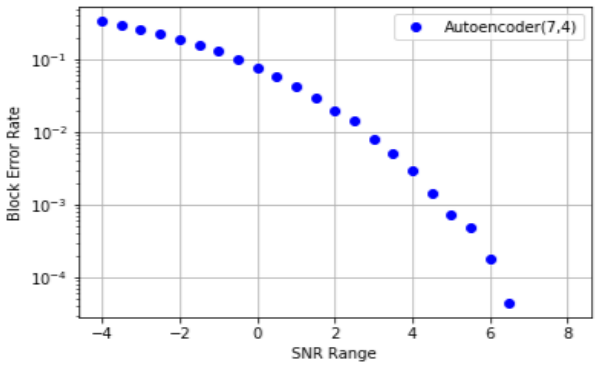

Figure 7.6: BLER versus E_b/N_0 for Autoencoder $(7, 4)$

saddle points during the training iterations.

It is a very novel concept to implement a communication system only by neural networks. But lots of researchers and engineers have worked on designing communication systems by expert knowledge for decades. If the autoencoder for communication system doesn't have great advantage, engineers won't apply it to design system. To do research about autoencoder for communication system, it only has academic meaning right now. To compare the traditional communication systems and autoencoders, we plot block error rate of uncoded BPSK (2,2) and autoencoder (2,2), and the BLER of QPSK in Figure. 7.7.

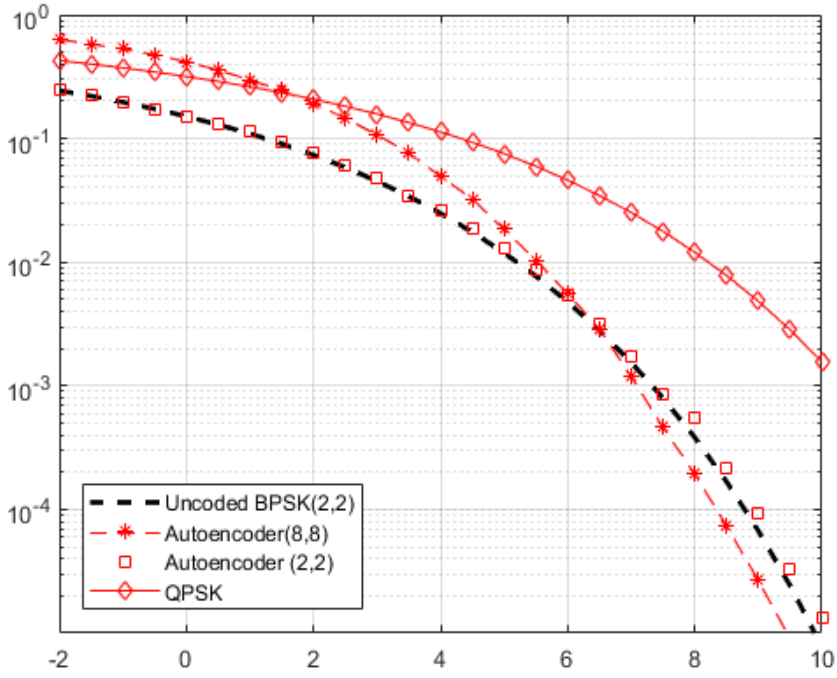

Figure 7.7: BLER versus E_b/N_0 for different autoencoders and traditional systems

Although in [1], the authors stated the constellation diagram the autoencoder for a $(2, 2)$ system is alike to QPSK constellation, we could not take QPSK to compare with autoencoder $(2, 2)$, because these two communication systems have no fair comparability. The reason is that autoencoder

$(2, 2)$ has two channels to transmit two bits for one time, whereas QPSK system only has one channel to transmit two bits for one time. The efficiency of QPSK is higher than the autoencoder $(2, 2)$. In addition, autoencoder $(2, 2)$ has learned how to encode and modulate through training, whereas QPSK is only a communication modulation scheme. To make comparison between autoencoder and traditional communication system, we take uncoded BPSK $(2, 2)$ and autoencoder (2,2) to compare their BLER at different values of E_b/N_0. From Figure. 7.7, it shows that the autoencoder $(2, 2)$, the most basic autoencoder for communication, could learn how to transmit messages and achieve the same performance as uncoded BPSK $(2, 2)$. Surprisingly, the autoencoder (8,8) outperforms the autoencoder $(2, 2)$ when the E_b/N_0 is larger than 6.5 dB. We could conclude that the autoencoder $(8, 8)$ has learned some joint modulation and coding scheme. In this way, some coding and modulation gains could be achieved by training the neural networks. This is a very exciting result. It means that if we increase the value of n, which is the number of channel we would use to transmit messages, even the communication R (n/k) is the same, we might get the joint coding and modulation gains. This is an advantage of applying deep learning to communication systems, as long as we figure out how to represent representations with high dimensions and then analyse these representations in a correct way.

Other detailed performance comparison for various parameters (n, k) would be done in the further research, because it took too much time to reproduce the results above, and the author of this thesis doesn't have enough time to do other works.

7.2 Future Work

We have tried to reproduce three parts of [1] but failed. These things are simulating autoencoder $(2, 4)$ with power constraint, autoencoders for multiple transmitters and receivers, and Radio Transformer Networks (RTN). In this section, we would describe the details about the difficulty of reproducing those results, and hope this section could be helpful for other researchers to finish this work. The authors in [1] have given a website about the source code of their results, but this website is not found right now.

To simulate autoencoder $(2, 4)$ with average power constraint, we build the neural networks by the following code:

```
input_signal=Input(shape=(M,))
encoded_signal = Dense(M, activation='relu')(input_signal)
```

```
encoded_signal_1 = Dense(n_channel, activation='linear')(
    encoded_signal)
encoded_signal_2 = BatchNormalization()(encoded_signal_1)

Eb_No_train = 5.01187 # coverted 7 db of EbNo
encoded_signal_3 = GaussianNoise(np.sqrt(1/(2*R*Eb_No_train)))(
    encoded_signal_2)

decoded_signal = Dense(M, activation='relu')(encoded_signal_3)
decoded_signal_1 = Dense(M, activation='softmax')(decoded_signal)
autoencoder_24 = Model(input_signal, decoded_signal_1)
adam = Adam(lr=0.01)
autoencoder_24.compile(optimizer=adam, loss='
    categorical_crossentropy')
```

To implement power constraint, we use the code

```
encoded2 = BatchNormalization()(encoded1)
```

to normalize the output vectors of encoded1 . And the constellation diagram is shown in Figure. 7.8.

This is not what we expect. We fail to reproduce the constellation diagram of autoencoder $(2, 4)$ in [1]. The constellation diagram of autoencoder $(2, 4)$ in [1] is shown in Figure. 7.9.

In Figure. 7.8∼7.9, these two figures show that the structure of Figure. 7.8 is the same as Figure. 7.9, but their amplitudes of the vectors are different. We suppose that the operation of BatchNormalization is useful to simulate the average power constraint, but we need to add other operations to modify the amplitude.

To simulate autoencoders for multiple transmitters and receivers, we use the following code to build two autoencoders. The parameters of these two autoencoders are both $(2, 2)$.

```
# Embedding Model for Two User using real signal
#user1's transmitter
u1_input_signal = Input(shape=(1,))
u1_encoded = embeddings.Embedding(input_dim=M, output_dim=emb_k,
    input_length=1)(u1_input_signal)
u1_encoded1 = Flatten()(u1_encoded)
u1_encoded2 = Dense(M, activation= 'relu')(u1_encoded1)
u1_encoded3 = Dense(n_channel, activation= 'linear')(u1_encoded2)
u1_encoded4 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.
    l2_normalize(x, dim=1))(u1_encoded3)
```

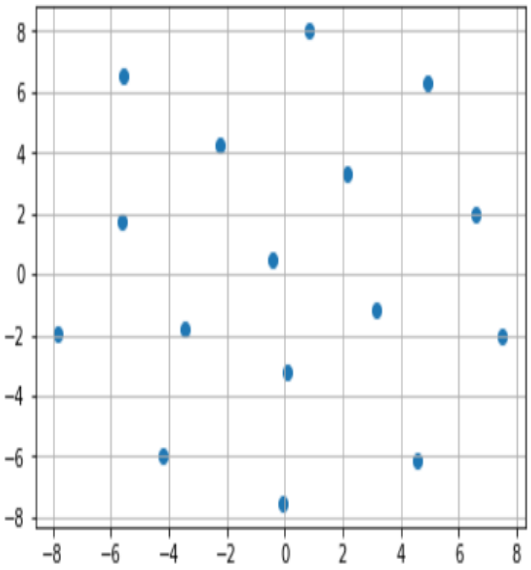

Figure 7.8: The constellation diagram of the failed simulation of autoencoder $(2, 4)$ with power constraint

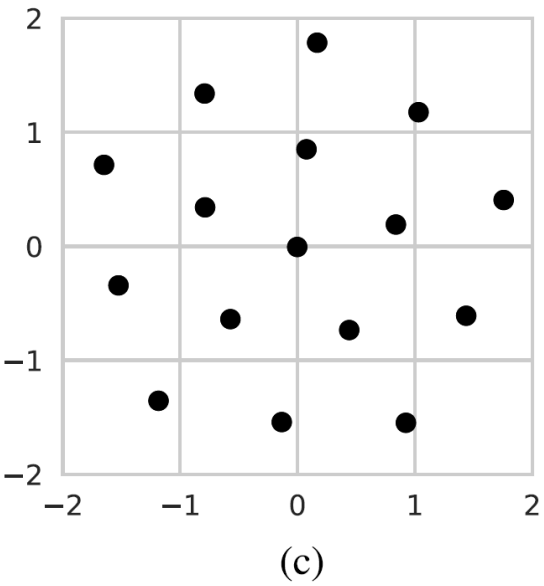

Figure 7.9: The constellation diagram of autoencoder $(2, 4)$ with power

```
#u1_encoded4 = BatchNormalization(momentum=0, center=False, scale=
    False)(u1_encoded3)
#user2's transmitter
u2_input_signal = Input(shape=(1,))
u2_encoded = embeddings.Embedding(input_dim=M, output_dim=emb_k,
    input_length=1)(u2_input_signal)
u2_encoded1 = Flatten()(u2_encoded)
u2_encoded2 = Dense(M, activation='relu')(u2_encoded1)
u2_encoded3 = Dense(n_channel, activation='linear')(u2_encoded2)
u2_encoded4 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.
    l2_normalize(x, dim=1))(u2_encoded3)
#u2_encoded4 = BatchNormalization(momentum=0, center=False, scale=
    False)(u2_encoded3)

#mixed AWGN channel
u1_channel_out = Lambda(lambda x: mixed_AWGN(x))([u1_encoded4,
    u2_encoded4])
u2_channel_out = Lambda(lambda x: mixed_AWGN(x))([u2_encoded4,
    u1_encoded4])

#user1's receiver
u1_decoded = Dense(M, activation='relu',name='u1_pre_receiver')(
    u1_channel_out)
u1_decoded1 = Dense(M, activation='softmax', name='u1_receiver')
    (u1_decoded)

#user2's receiver
u2_decoded = Dense(M, activation='relu',name='u2_pre_receiver')(
    u2_channel_out)
u2_decoded1 = Dense(M, activation='softmax',name='u2_receiver')(
    u2_decoded)

twouser_autoencoder = Model(inputs=[u1_input_signal,
    u2_input_signal],
outputs=[u1_decoded1, u2_decoded1])
adam =Adam(lr = 0.01)
twouser_autoencoder.compile( optimizer=adam,
loss='sparse_categorical_crossentropy',
loss_weights=[alpha, beta])#loss=a*loss1+b*loss2
```

However, the result is shown in Figure. 7.10 and this is not what we expect. The constellation diagram of [1] for two users with parameters $(2, 2)$ is shown in Figure. 7.11. It could be seen that the simulation result is toally different the result in [1].

The concept of radio transformer networks (RTN) for augmented signal processing algorithms is modifying the structure of receiver of the au-

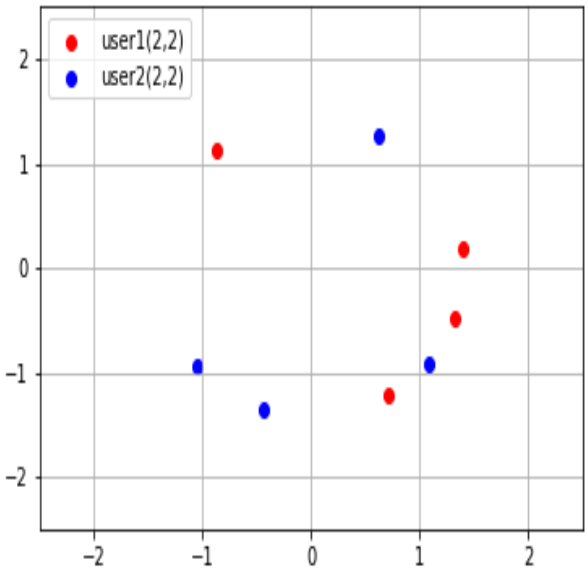

Figure 7.10: The simulation result of constellation diagram for two-user with parameters $(2, 2)$ (not expected)

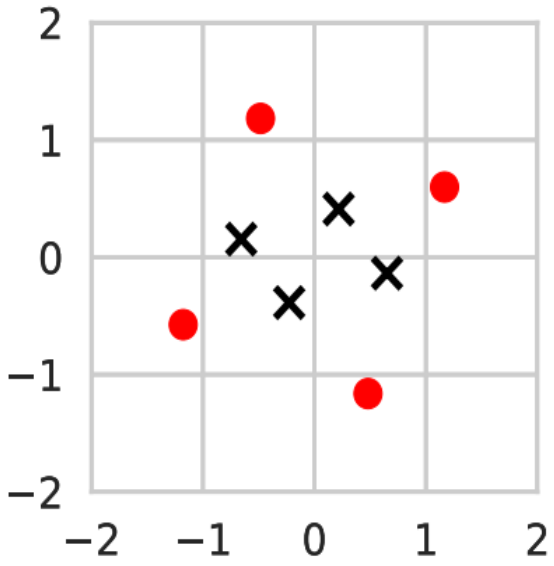

Figure 7.11: The constellation diagram [1] for two-user with parameters $(2, 2)$

toencoder to improve the performance of the decoder and get lower value of BLER. The structure of RTN is shown in Figure. 7.12. The figure is from [38].

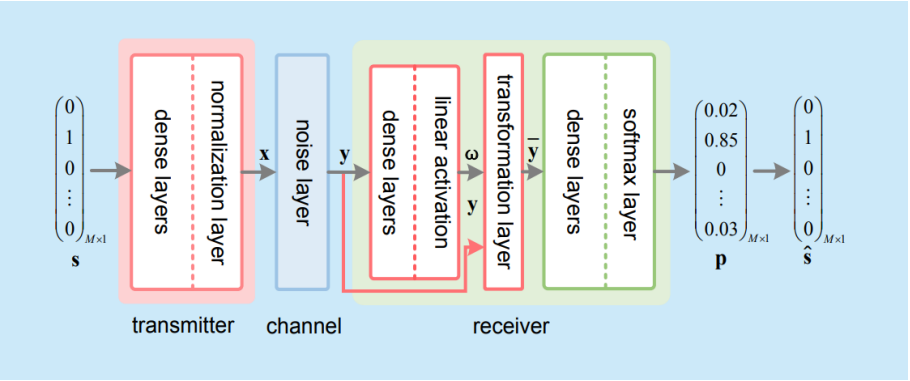

Figure 7.12: A receiver of an autoencoder represented as an RTN

The authors in [1] stated RTNs are inspired by Spatial Transformer Networks (STNs) [41]. In RTN, the operation of adding the transformation layer is not included API of Keras and other deep learning libraries. Thus, to simulate RTN and test the performance of it, we need to write functions about RTN following the instructions in [1] and [41]. It is out of the scope of this thesis and we would leave that to future researches.

Chapter 8

Conclusion

We have discussed the novel application of deep learning to the communication systems. Most importantly, we reproduce antoencoders for communication systems and evaluate the performance of these. We use Keras to generate the datasets, and build autoencoders with different parameters.

From the results, we could conclude that autoencoders could achieve competitive BLER performance compared with traditional communication systems. However, there is a problem. To train and represent autoencoder with higher dimension remain a challenge.

One advantage of DL-based communication system is the autoencoder could learn how to communicate even if the optimal schemes are unknown (e.g., in scenarios that we don't know how to represent the channel by rigid mathematics). In addition, the autoencoders for communication systems have excellent expressive capacity and convenient optimization. As long as we figure out how to represent the output vectors of the transmitter and have the effective analysis methodology, the DL-based communication system might yield significant improvements on the physical layer. For now, it is a promising research area and far from maturity. A wide range of studies into DL for physical layer is needed to be done, and we believe researchers would conduct further studies in this promising area, including theoretical analysis, implementation in real-world scenarios.

How to implement this novel concept of designing communication system as an autoencoder in the reality is out of the scope of this thesis and left to future investigations. During the analysis of the results, the most difficult thing is to get the representations of the output vector at the transmitter side. As for the future work, an important issue to consider is to apply more efficient dimensionality reduction methodologies to analyse to generate the

constellation diagrams. The other big issue is we still don't know that how the DL-based models learn to modulate and encode, and it would cause difficulties to improve the DL-based communication systems.

Bibliography

[1] T. J. O'Shea and J. Hoydis. (2017) An introduction to deep learning for the physical layer. [Online]. Available: https://arxiv.org/abs/1702.00832, preprint.

[2] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function [J]. Mathematics of Control, Signals, and Systems, 1989, 2: 303-314.

[3] G. Tesauro. Practical Issues in Temporal Difference Learning [J]. Machine Learning, 1992, 8: 257-277.

[4] A.Y. Ng, N. Jiquan, Y.F. Chuan, M. Yifan, and S. Caroline. Unsupervised Feature Learning and Deep Learning Tutorial, 2013.

[5] G.E. Hinton and R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks [J]. Science, 2006, 313(5786): 504-507.

[6] G.E. Hinton, S. Osindero, and Y.W. Teh. A Fast Learning Algorithm for Deep Belief Nets [J]. Neural Computation, 2006, 18(7): 1527-1554.

[7] Y. Jia et al., "Caffe: Convolutional architecture for fast feature embedding," in Proc. 22nd ACM Int. Conf. Multimedia, Orlando, FL, USA, 2014, pp. 675–678.

[8] T. Chen et al., "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," arXiv preprint arXiv:1512.01274, 2015.

[9] M. Abadi et al. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. [Online]. Available: http://tensorflow.org/

[10] R. Al-Rfou et al., "Theano: A Python framework for fast computation of mathematical expressions," arXiv preprint arXiv:1605.02688, 2016.

[11] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A MATLAB like environment for machine learning," in Proc. BigLearn NIPS Workshop, 2011, pp. 1–6.

[12] F. Chollet. (2015). Keras. [Online]. Available: https://github.com/fchollet/keras

[13] Y. LeCun, "Generalization and network design strategies," in Connectionism in Perspective. Amsterdam, The Netherlands: North-Holland, 1989, pp. 143–155.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in Proc. IEEE Int. Conf. Comput. Vis., Santiago, Chile, 2015, pp. 1026–1034.

[15] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, "Deep learning for identifying metastatic breast cancer," arXiv preprint arXiv:1606.05718, 2016.

[16] D. George and E. A. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," arXiv preprint arXiv:1701.00008, 2016.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[18] R. Sommer, V. Paxson, "Outside the closed world: on using machine learning for network intrusion detection" //Proceedings of the 2010 IEEE Symposium on Security and Privacy. Washington, USA, 2010: 305-316

[19] A. Buczak, E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection" IEEE Communication Surveys & Tutorials, 2016, 18(2): 1153-1176

[20] J. Cannady, "Artificial neural networks for misuse detection," //Proceedings of the 1998 National Information Systems Security Conference. Arlington, USA, 1998: 443-456

[21] M. Ibnkahla, "Applications of neural networks to digital communications—A survey," Elsevier Signal Process., vol. 80, no. 7, pp. 1185–1215, 2000.

[22] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," IEEE Commun. Surveys Tuts., vol. 15, no. 3, pp. 1136–1159, 3rd Quart., 2013.

[23] S. D¨orner, S. Cammerer, J. Hoydis, and S. ten Brink. Deep Learning Based Communication Over the Air. ArXiv e-prints, July 2017. Mathematics of Control, Signals, and Systems, 1989, 2: 303-314.

[24] N. Farsad and A. Goldsmith, "Detection algorithms for communication systems using deep learning," arXiv preprint arXiv:1705.08044, 2017.

[25] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in Proc. Int. Conf. Eng. Appl. Neural Netw., Aberdeen, U.K., 2016, pp. 213–226.

[26] M. Borgerding and P. Schniter, "Onsager-corrected deep learning for sparse linear inverse problems," arXiv preprint arXiv:1607.05966, 2016.

[27] A. Mousavi and R. G. Baraniuk, "Learning to invert: Signal recovery via deep convolutional networks," arXiv preprint arXiv:1701.03891, 2017.

[28] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," arXiv preprint arXiv:1610.06918, 2016.

[29] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[30] A. Mousavi and R. G. Baraniuk, "Learning to invert: Signal recovery via deep convolutional networks," arXiv preprint arXiv:1701.03891, 2017.

[31] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," arXiv preprint arXiv:1610.06918, 2016.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proc. Int. Conf. Learn. Represent. (ICLR), San Diego, CA, USA, 2015, pp. 1–15.

[33] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," J. Mach. Learn. Res., vol. 9, pp. 2579–2605, Nov. 2008.

[34] T. S. Rappaport, Wireless communications: Principles and practice, 2nd ed. Prentice Hall, 2002.

[35] D. Erhan, Y. Bengio, A.C. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does Unsupervised Pre-training Help Deep Learning [J]. Machine Learning Research, 2010, 11: 625-660.

[36] D. Erhan, P.-A. Manzagol, Y. Bengio, and P. Vincent. The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training [C]. Conference on Artificial Intelligence and Statistics (AISTATS), 2009, 5: 153-160.

[37] G.E. Hinton, S. Osindero, and Y.W. Teh. A Fast Learning Algorithm for Deep Belief Nets [J]. Neural Computation, 2006, 18(7): 1527-1554.

[38] T. Wang, C.-K. Wen, H. Wang, F. Gao, T. Jiang, and S. Jin, ‘‘Deep learning for wireless physical layer: Opportunities and challenges,’’ China Commun., vol. 14, no. 11, pp. 92–111, 2017.

[39] “Anaconda,” https://www.anaconda.com/download/

[40] Y. Wang and K. Xu, “source code and simulation results,” https://github.com/asueeer/Source-code-and-simulation-results

[41] M. Jaderberg, K. Simonyan, A. Zisserman et al., “Spatial transformer networks,” in Advances in Neural Information Processing Systems (NIPS), 2015, pp. 2017–2025.