Progress Report of Final Project

Module Name	17ELP005 - Project in Networked Communications
Name	Yisu Wang
ID Number	B633493

Progress Report of Final Project

Abstract.

This progress report is to list some preliminary work of the project of networked communications. The work includes methodology and some preliminary results.
The project is inspired by a paper which is [1]. The project would mainly reproduce the results of this paper and extend those results.

1. Literature Review

Since very recently, open-source DL software libraries (e.g., Caffe [2], MXNet [3], TensorFlow [4], Theano [5], Torch [6], Keras [7]), and powerful specialized hardware, such as field programmable gate arrays (FPGAs) and processing units (GPUs) are cheaply and readily available. Thanks to these rapid developments, the applications of DL are applied to almost every research domain [8-12]. Especially, DL shines in domains such as computer vision (CV) and natural language processing (NLP), which are difficult to characterize practical tasks with rigid mathematical models.

In communications, researchers have tried to extend machine learning (ML) towards communications in the past, but they mainly focus on cyberspace security [13-15].

Although several researchers have also addressed problems related to physical layer with ML such as channel modelling and prediction, equalization, quantization [16-17], etc., ML did not cause any fundamental impact on the physical layer. The main reason for this 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 characterize real effects, we could design extremely accurate communication systems that enable robust algorithms for symbol detection.

However, [1] presents 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 DL could be useful tools applied to improve current wireless communications. And when channel models are difficult to derive, researchers could turn to DL from traditional signal processing algorithms to deduce the channel.

Based on these ideas from [1], [18] implements a communication system using only deep neural networks by software-define-radios (SDRs). The results from [18] demonstrate that the autoencoder idea could be implemented in the reality. To implement this fascinating novel antoencoder concept using SDRs, [1] extends the existing concepts toward continuous signal transmission, which entails the receiver synchronization issue. [18] overcomes 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 [19], modulation recognition [20], compressed sensing [21], [22], learning of encryption/decryption schemes for an eavesdropper channel [23]. There are two main different viewpoints of applying DL to the communication systems in these papers. The goal is to either completely replace existing communication algorithms with DL,

or to apply DL only for improving/augmenting them.

2. DEEP LEARNING BASICS

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

$$r_l = f(r_{l-1};\theta_l),\quad l=1,\dots,L \tag{1}$$

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

$$f(r_{l-1};\theta_l)=\sigma(\mathrm{W}_l r_{l-1}+\mathrm{b}_l) \tag{2}$$

the lth layer is called *dense* or *full-connected* layer, where $\mathrm{W}_l\in\mathbb{R}^{N_l}\times\mathbb{R}^{N_{l-1}},\mathrm{b}_l\in\mathbb{R}^{N_l}$, and $\sigma(\cdot)$ is an *activation* function, and $\theta_l=\{\mathrm{W}_l,\mathrm{b}_l\}$. Common activation functions and layer types are listed in *Table. 1* and *Table. 2* respectively.

TABLE I
LIST OF LAYER TYPES

Name	$f_\ell(\mathbf{r}_{\ell-1};\theta_\ell)$	θ_ℓ
Dense	$\sigma\left(\mathbf{W}_\ell\mathbf{r}_{\ell-1}+\mathbf{b}_\ell\right)$	$\mathbf{W}_\ell, \mathbf{b}_\ell$
Noise	$\mathbf{r}_{\ell-1}+\mathbf{n},\ \mathbf{n}\sim\mathcal{N}(\mathbf{0},\beta\mathbf{I}_{N_{\ell-1}})$	none
Dropout [36]	$\mathbf{d}\odot\mathbf{r}_{\ell-1},\ d_i\sim\mathrm{Bern}(\alpha)$	none
Normalization	e.g., $\frac{\sqrt{N_{\ell-1}}\mathbf{r}_{\ell-1}}{\|\mathbf{r}_{\ell-1}\|_2}$	none

TABLE II
LIST OF ACTIVATION FUNCTIONS

Name	$[\sigma(\mathbf{u})]_i$	Range
linear[1]	u_i	$(-\infty,\infty)$
ReLU [37]	$\max(0,u_i)$	$[0,\infty)$
tanh	$\tanh(u_i)$	$(-1,1)$
sigmoid	$\frac{1}{1+e^{-u_i}}$	$(0,1)$
softmax	$\frac{e^{u_i}}{\sum_j e^{u_j}}$	$(0,1)$

TABLE III
LIST OF LOSS FUNCTIONS

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

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

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

$$L(\theta) = \frac{1}{S}\sum_{i=1}^{S} l\left(r^*_{L,i}, r_{L,i}\right) \tag{3}$$

Where $l(u,v)\colon \mathbb{R}^{N_L}\times\mathbb{R}^{N_L}\mapsto\mathbb{R}$ is the loss function, which is categorical cross-entropy function $-\sum_j u_j\, log(v_j)$; and $r_{L,i}$ is the output and we use $r_{0,i}$ as input. Several commonly used loss functions are presented in *Table.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

$$\theta_{t+1} = \theta_t - \eta\nabla\tilde{L}(\theta_t) \tag{4}$$

Where the learning rate $\eta>0$, and $\tilde{L}(\theta_t)$ is the approximation of the categorical cross-entropy function.

Note that detailed description of Neural Networks is presented in [24].

This paper defines and trains NNs by the existing DL libraries presented in Literature Review. To simulate autoencoder concept from [1], this work mainly uses TesnsorFlow [4] and Keras [7].

3. Preliminary Results

This paper reproduces some results of [1]. The code is listed in the Appendix. Some meaningful results would be listed below. And the analysis of those results would be written in the final report.

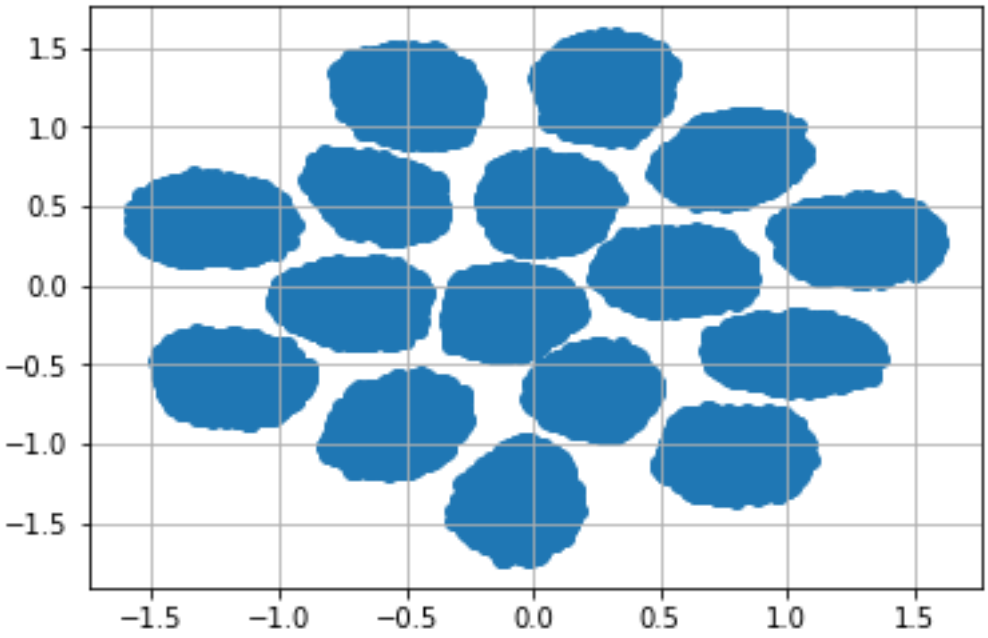

Fig. 1 Consellation(7,4)

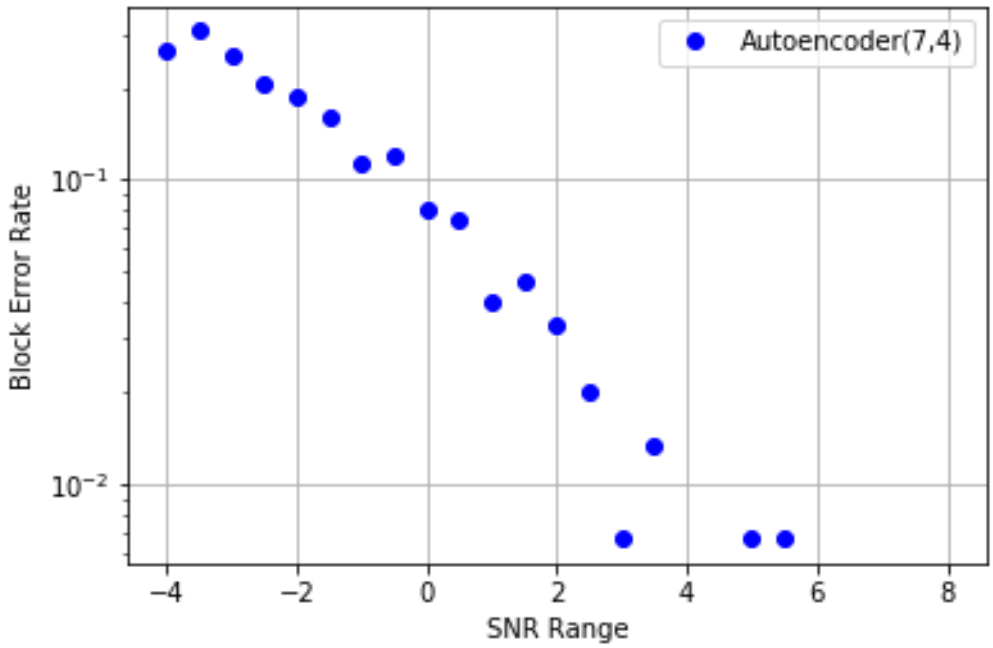

Fig. 2 AutoEncoder(7,4)_BER

Reference

[1] Timothy J. O'Shea and Jakob Hoydis. An introduction to machine learning communications systems. CoRR, abs/1702.00832, 2017.

[2] 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.

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

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

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

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

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

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

[9] 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.

[10] 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*.

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

[12] 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.

[13] 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

[14] 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

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

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

[17] 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.

[18] S. D ̈orner, S. Cammerer, J. Hoydis, and S. ten Brink. Deep Learning Based Communication Over the Air. ArXiv e-prints, July 2017.

[19] Nariman Farsad and Andrea J. Goldsmith. Neural network detection of data sequences in

communication systems. 2018

[20] 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.

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

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

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

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

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

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

Reference

Code

1. Autoencoder2_2

```
# importing libs# impor
import numpy as np
import tensorflow as tf
import keras
from keras.layers import Input, Dense, GaussianNoise,Lambda,Dropout
from keras.models import Model
from keras import regularizers
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam,SGD
from keras import backend as K
# for reproducing reslut
from numpy.random import seed
seed(1)
from tensorflow import set_random_seed
set_random_seed(3)
# defining parameters
# define (n,k) here for (n,k) autoencoder
# n = n_channel
# k = log2(M)  ==> so for (7,4) autoencoder n_channel = 7 and M = 2^4 = 16
M = 4
k = np.log2(M)
k = int(k)
n_channel = 2
R = k/n_channel
print ('M:',M,'k:',k,'n:',n_channel)
#generating data of size N#genera
N = 8000
label = np.random.randint(M,size=N)
# creating one hot encoded vectors
data = []
for i in label:
    temp = np.zeros(M)
    temp[i] = 1
    data.append(temp)

# checking data shape
```

```
data = np.array(data)
print (data.shape)
# checking generated data with it's label
temp_check = [17,23,45,67,89,96,72,250,350]
for i in temp_check:
    print(label[i],data[i])


# defining autoencoder and it's layer
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)
#encoded2 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.l2_normalize(x, dim=1))(encoded1)#这里修改了作者
的代码！！！不过应该是一样的
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')
# printing summary of layers and it's trainable parameters
print (autoencoder.summary())
# for tensor board visualization
#tbCallBack = keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0, batch_size=32,
write_graph=True, write_grads=True, write_images=False, embeddings_freq=0,
embeddings_layer_names=None, embeddings_metadata=None)
# traning auto encoder# trani
autoencoder.fit(data, data,
                epochs=45,
                batch_size=32)
# saving keras model
from keras.models import load_model
# if you want to save model then remove below comment
# autoencoder.save('autoencoder_v_best.model')
# making encoder from full autoencoder
encoder = Model(input_signal, encoded2)
# making decoder from full autoencoder
encoded_input = Input(shape=(n_channel,))
deco = autoencoder.layers[-2](encoded_input)
deco = autoencoder.layers[-1](deco)
decoder = Model(encoded_input, deco)
# generating data for checking BER
# if you're not using t-sne for visulation than set N to 70,000 for better result
```

```
# for t-sne use less N like N = 1500
N = 50000
test_label = np.random.randint(M,size=N)
test_data = []
for i in test_label:
    temp = np.zeros(M)
    temp[i] = 1
    test_data.append(temp)

test_data = np.array(test_data)
# checking generated data
temp_test = 6
print (test_data[temp_test][test_label[temp_test]],test_label[temp_test])
# for plotting learned consteallation diagram
scatter_plot = []
for i in range(0,M):
    temp = np.zeros(M)
    temp[i] = 1
    scatter_plot.append(encoder.predict(np.expand_dims(temp,axis=0)))
scatter_plot = np.array(scatter_plot)
print (scatter_plot.shape)
# use this function for ploting constellation for higher dimenson like 7-D for (7,4) autoencoder # use
t
'''
x_emb = encoder.predict(test_data)
noise_std = np.sqrt(1/(2*R*EbNo_train))
noise = noise_std * np.random.randn(N,n_channel)
x_emb = x_emb + noise
from sklearn.manifold import TSNE
X_embedded = TSNE(learning_rate=700, n_components=2,n_iter=35000, random_state=0,
perplexity=60).fit_transform(x_emb)
print (X_embedded.shape)
X_embedded = X_embedded / 7
import matplotlib.pyplot as plt
plt.scatter(X_embedded[:,0],X_embedded[:,1])
#plt.axis((-2.5,2.5,-2.5,2.5))
plt.grid()
plt.show()
'''
# ploting constellation diagram
import matplotlib.pyplot as plt
scatter_plot = scatter_plot.reshape(M,2,1)
plt.scatter(scatter_plot[:,0],scatter_plot[:,1])
plt.axis((-2.5,2.5,-2.5,2.5))
```

```
plt.grid()
plt.show()
def frange(x, y, jump):
  while x < y:
    yield x
    x += jump

# calculating BER
# this is optimized BER function so it can handle large number of N
# previous code has another for loop which was making it slow
EbNodB_range = list(frange(-4,8.5,0.5))
ber = [None]*len(EbNodB_range)
for n in range(0,len(EbNodB_range)):
    EbNo=10.0**(EbNodB_range[n]/10.0)
    noise_std = np.sqrt(1/(2*R*EbNo))
    noise_mean = 0
    no_errors = 0
    nn = N
    noise = noise_std * np.random.randn(nn,n_channel)
    encoded_signal = encoder.predict(test_data)
    final_signal = encoded_signal + noise
    pred_final_signal =  decoder.predict(final_signal)
    pred_output = np.argmax(pred_final_signal,axis=1)
    no_errors = (pred_output != test_label)
    no_errors =  no_errors.astype(int).sum()
    ber[n] = no_errors / nn
    print ('SNR:',EbNodB_range[n],'BER:',ber[n])
    # use below line for generating matlab like matrix which can be copy and paste for plotting ber graph
in matlab
    #print(ber[n], " ",end='')

# ploting ber curve
import matplotlib.pyplot as plt
from scipy import interpolate
plt.plot(EbNodB_range, ber, 'bo',label='Autoencoder(2,2)')
plt.yscale('log')
plt.xlabel('SNR Range')
plt.ylabel('Block Error Rate')
plt.grid()
plt.legend(loc='upper right',ncol = 1)
# for saving figure remove below comment# for s
#plt.savefig('AutoEncoder_2_2_constrained_BER_matplotlib')
plt.show()
```

2. Autoencoder2_4 power constraint

```
# importing libs# impor
import numpy as np
import tensorflow as tf
import keras
from keras.layers import Input, Dense, GaussianNoise,Lambda,Dropout
from keras.models import Model
from keras import regularizers
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam,SGD
from keras import backend as K
# for reproducing reslut
from numpy.random import seed
seed(1)
from tensorflow import set_random_seed
set_random_seed(3)
# defining parameters
# define (n,k) here for (n,k) autoencoder
# n = n_channel
# k = log2(M)  ==> so for (7,4) autoencoder n_channel = 7 and M = 2^4 = 16
M = 16
k = np.log2(M)
k = int(k)
n_channel = 2
R = k/n_channel
print ('M:',M,'k:',k,'n:',n_channel)
#generating data of size N#genera
N = 800000
label = np.random.randint(M,size=N)
# creating one hot encoded vectors
data = []
for i in label:
    temp = np.zeros(M)
    temp[i] = 1
    data.append(temp)

# checking data shape
data = np.array(data)
print (data.shape)
# checking generated data with it's label
temp_check = [17,23,45,67,89,96,72,250,350]
```

```
for i in temp_check:
    print(label[i],data[i])


# defining autoencoder and it's layer
input_signal = Input(shape=(M,))
encoded = Dense(M, activation='relu')(input_signal)
encoded1 = Dense(n_channel, activation='linear')(encoded)
encoded2 = BatchNormalization()(encoded1)
#encoded2 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.l2_normalize(x, dim=1))(encoded1)
#encoded2 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.l2_normalize(x, dim=1))(encoded1)#这里修改了作者
的代码！！！不过应该是一样的
#encoded2 = Lambda(lambda x: np.sqrt(n_channel)*K.l2_normalize(x,axis=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')
# printing summary of layers and it's trainable parameters
print (autoencoder.summary())
# for tensor board visualization
#tbCallBack = keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0, batch_size=32,
write_graph=True, write_grads=True, write_images=False, embeddings_freq=0,
embeddings_layer_names=None, embeddings_metadata=None)
# traning auto encoder# trani
autoencoder.fit(data, data,
                epochs=45,
                batch_size=32)
# saving keras model
from keras.models import load_model
# if you want to save model then remove below comment
# autoencoder.save('autoencoder_v_best.model')
# making encoder from full autoencoder
encoder = Model(input_signal, encoded2)
# making decoder from full autoencoder
encoded_input = Input(shape=(n_channel,))
deco = autoencoder.layers[-2](encoded_input)
deco = autoencoder.layers[-1](deco)
decoder = Model(encoded_input, deco)
# generating data for checking BER
# if you're not using t-sne for visulation than set N to 70,000 for better result
# for t-sne use less N like N = 1500
N = 50000
```

```
test_label = np.random.randint(M,size=N)
test_data = []
for i in test_label:
    temp = np.zeros(M)
    temp[i] = 1
    test_data.append(temp)

test_data = np.array(test_data)
# checking generated data
temp_test = 6
print (test_data[temp_test][test_label[temp_test]],test_label[temp_test])
# for plotting learned consteallation diagram
scatter_plot = []
for i in range(0,M):
    temp = np.zeros(M)
    temp[i] = 1
    scatter_plot.append(encoder.predict(np.expand_dims(temp,axis=0)))
scatter_plot = np.array(scatter_plot)
print (scatter_plot.shape)
# ploting constellation diagram
# use this function for ploting constellation for higher dimenson like 7-D for (7,4) autoencoder # use
t
#x_emb = encoder.predict(test_data)
#noise_std = np.sqrt(1/(2*R*EbNo_train))
#noise = noise_std * np.random.randn(N,n_channel)
#x_emb = x_emb + noise
#from sklearn.manifold import TSNE
#X_embedded = TSNE(learning_rate=700, n_components=2,n_iter=35000, random_state=0,
perplexity=60).fit_transform(x_emb)
#print (X_embedded.shape)
#X_embedded = X_embedded / 7
#import matplotlib.pyplot as plt
#plt.scatter(X_embedded[:,0],X_embedded[:,1])
##plt.axis((-2.5,2.5,-2.5,2.5))
#plt.grid()
#plt.show()
# ploting constellation diagram
import matplotlib.pyplot as plt
scatter_plot = scatter_plot.reshape(M,2,1)
plt.scatter(scatter_plot[:,0],scatter_plot[:,1])
#plt.axis((-2.5,2.5,-2.5,2.5))
plt.grid()
plt.show()
def frange(x, y, jump):
```

```
  while x < y:
    yield x
    x += jump

# calculating BER
# this is optimized BER function so it can handle large number of N
# previous code has another for loop which was making it slow
EbNodB_range = list(frange(-4,8.5,0.5))
ber = [None]*len(EbNodB_range)
for n in range(0,len(EbNodB_range)):
    EbNo=10.0**(EbNodB_range[n]/10.0)
    noise_std = np.sqrt(1/(2*R*EbNo))
    noise_mean = 0
    no_errors = 0
    nn = N
    noise = noise_std * np.random.randn(nn,n_channel)
    encoded_signal = encoder.predict(test_data)
    final_signal = encoded_signal + noise
    pred_final_signal =  decoder.predict(final_signal)
    pred_output = np.argmax(pred_final_signal,axis=1)
    no_errors = (pred_output != test_label)
    no_errors =  no_errors.astype(int).sum()
    ber[n] = no_errors / nn
    print ('SNR:',EbNodB_range[n],'BER:',ber[n])
    # use below line for generating matlab like matrix which can be copy and paste for plotting ber graph
in matlab
    #print(ber[n], " ",end='')

# ploting ber curve
import matplotlib.pyplot as plt
from scipy import interpolate
plt.plot(EbNodB_range, ber, 'bo',label='Autoencoder(2,2)')
plt.yscale('log')
plt.xlabel('SNR Range')
plt.ylabel('Block Error Rate')
plt.grid()
plt.legend(loc='upper right',ncol = 1)
# for saving figure remove below comment# for s
#plt.savefig('AutoEncoder_2_2_constrained_BER_matplotlib')
plt.show()
```

3. Autoencoder2_4

```
import numpy as np
import tensorflow as tf
import keras
from keras.layers import Input, Dense, GaussianNoise,Lambda,Dropout
from keras.models import Model
from keras import regularizers
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam,SGD
from keras import backend as K
# for reproducing reslut
from numpy.random import seed
seed(1)
from tensorflow import set_random_seed
set_random_seed(3)
# defining parameters
# define (n,k) here for (n,k) autoencoder
# n = n_channel
# k = log2(M)  ==> so for (7,4) autoencoder n_channel = 7 and M = 2^4 = 16
M = 16
k = np.log2(M)
k = int(k)
n_channel = 2
R = k/n_channel
print ('M:',M,'k:',k,'n:',n_channel)
#generating data of size N#genera
N = 8000
label = np.random.randint(M,size=N)
# creating one hot encoded vectors
data = []
for i in label:
    temp = np.zeros(M)
    temp[i] = 1
    data.append(temp)

# checking data shape
data = np.array(data)
print (data.shape)
# checking generated data with it's label
temp_check = [17,23,45,67,89,96,72,250,350]
for i in temp_check:
```

```
    print(label[i],data[i])


# defining autoencoder and it's layer
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)
#encoded2 = Lambda(lambda x: np.sqrt(n_channel)*tf.nn.l2_normalize(x, dim=1))(encoded1)#这里修改了作者的代码！！！不过应该是一样的
#encoded2 = Lambda(lambda x: np.sqrt(n_channel)*K.l2_normalize(x,axis=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')
# printing summary of layers and it's trainable parameters
print (autoencoder.summary())
# for tensor board visualization
#tbCallBack = keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0, batch_size=32, write_graph=True, write_grads=True, write_images=False, embeddings_freq=0, embeddings_layer_names=None, embeddings_metadata=None)
# traning auto encoder# trani
autoencoder.fit(data, data,
                epochs=45,
                batch_size=32)
# saving keras model
from keras.models import load_model
# if you want to save model then remove below comment
# autoencoder.save('autoencoder_v_best.model')
# making encoder from full autoencoder
encoder = Model(input_signal, encoded2)
# making decoder from full autoencoder
encoded_input = Input(shape=(n_channel,))
deco = autoencoder.layers[-2](encoded_input)
deco = autoencoder.layers[-1](deco)
decoder = Model(encoded_input, deco)
# generating data for checking BER
# if you're not using t-sne for visulation than set N to 70,000 for better result
# for t-sne use less N like N = 1500
N = 50000
test_label = np.random.randint(M,size=N)
test_data = []
```

```
for i in test_label:
    temp = np.zeros(M)
    temp[i] = 1
    test_data.append(temp)

test_data = np.array(test_data)
# checking generated data
temp_test = 6
print (test_data[temp_test][test_label[temp_test]],test_label[temp_test])
# for plotting learned consteallation diagram
scatter_plot = []
for i in range(0,M):
    temp = np.zeros(M)
    temp[i] = 1
    scatter_plot.append(encoder.predict(np.expand_dims(temp,axis=0)))
scatter_plot = np.array(scatter_plot)
print (scatter_plot.shape)
# ploting constellation diagram
# use this function for ploting constellation for higher dimenson like 7-D for (7,4) autoencoder
# use t
#x_emb = encoder.predict(test_data)
#noise_std = np.sqrt(1/(2*R*EbNo_train))
#noise = noise_std * np.random.randn(N,n_channel)
#x_emb = x_emb + noise
#from sklearn.manifold import TSNE
#X_embedded = TSNE(learning_rate=700, n_components=2,n_iter=35000, random_state=0,
perplexity=60).fit_transform(x_emb)
#print (X_embedded.shape)
#X_embedded = X_embedded / 7
#import matplotlib.pyplot as plt
#plt.scatter(X_embedded[:,0],X_embedded[:,1])
##plt.axis((-2.5,2.5,-2.5,2.5))
#plt.grid()
#plt.show()
# ploting constellation diagram
import matplotlib.pyplot as plt
scatter_plot = scatter_plot.reshape(M,2,1)
plt.scatter(scatter_plot[:,0],scatter_plot[:,1])
plt.axis((-2.5,2.5,-2.5,2.5))
plt.grid()
plt.show()
def frange(x, y, jump):
  while x < y:
    yield x
```

```
    x += jump

# calculating BER
# this is optimized BER function so it can handle large number of N
# previous code has another for loop which was making it slow
EbNodB_range = list(frange(-4,8.5,0.5))
ber = [None]*len(EbNodB_range)
for n in range(0,len(EbNodB_range)):
    EbNo=10.0**(EbNodB_range[n]/10.0)
    noise_std = np.sqrt(1/(2*R*EbNo))
    noise_mean = 0
    no_errors = 0
    nn = N
    noise = noise_std * np.random.randn(nn,n_channel)
    encoded_signal = encoder.predict(test_data)
    final_signal = encoded_signal + noise
    pred_final_signal =  decoder.predict(final_signal)
    pred_output = np.argmax(pred_final_signal,axis=1)
    no_errors = (pred_output != test_label)
    no_errors =  no_errors.astype(int).sum()
    ber[n] = no_errors / nn
    print ('SNR:',EbNodB_range[n],'BER:',ber[n])
    # use below line for generating matlab like matrix which can be copy and paste for plotting ber
graph in matlab
    #print(ber[n], " ",end='')

# ploting ber curve
import matplotlib.pyplot as plt
from scipy import interpolate
plt.plot(EbNodB_range, ber, 'bo',label='Autoencoder(2,2)')
plt.yscale('log')
plt.xlabel('SNR Range')
plt.ylabel('Block Error Rate')
plt.grid()
plt.legend(loc='upper right',ncol = 1)
# for saving figure remove below comment# for s
#plt.savefig('AutoEncoder_2_2_constrained_BER_matplotlib')
plt.show()
```

4. Autoencoder7_4

```
import numpy as np
import tensorflow as tf
from keras.layers import Input, Dense, GaussianNoise
from keras.layers.recurrent import LSTM
from keras.models import Model
from keras import regularizers
from keras.layers.normalization import BatchNormalization
from keras.optimizers import SGD
import random as rn
# defining parameters
M = 16
k = np.log2(M)
k = int(k)
print ('M:',M,'k:',k)
#generating data of size N
N = 10000
label = np.random.randint(M,size=N)
# creating one hot encoded vectors
data = []
for i in label:
    temp = np.zeros(M)
    temp[i] = 1
    data.append(temp)



data = np.array(data)
print (data.shape)
temp_check = [17,23,45,67,89,96,72,250,350]
for i in temp_check:
    print(label[i],data[i])



R = 4/7
n_channel = 7
print (int(k/R))
input_signal = Input(shape=(M,))
encoded = Dense(M, activation='relu')(input_signal)
encoded1 = Dense(n_channel, activation='linear')(encoded)
encoded2 = BatchNormalization()(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)
#sgd = SGD(lr=0.001)
autoencoder.compile(optimizer='adam', loss='categorical_crossentropy')
print (autoencoder.summary())
N_val = 1500
val_label = np.random.randint(M,size=N_val)
val_data = []
for i in val_label:
    temp = np.zeros(M)
    temp[i] = 1
    val_data.append(temp)
val_data = np.array(val_data)
autoencoder.fit(data, data,
                epochs=17,
                batch_size=300,
                validation_data=(val_data, val_data))
from keras.models import load_model
#autoencoder.save('4_7_symbol_autoencoder_v_best.model')
#autoencoder_loaded = load_model('4_7_symbol_autoencoder_v_best.model')
encoder = Model(input_signal, encoded2)
encoded_input = Input(shape=(n_channel,))
deco = autoencoder.layers[-2](encoded_input)
deco = autoencoder.layers[-1](deco)
# create the decoder model
decoder = Model(encoded_input, deco)
N = 45000
test_label = np.random.randint(M,size=N)
test_data = []
for i in test_label:
    temp = np.zeros(M)
    temp[i] = 1
    test_data.append(temp)


test_data = np.array(test_data)
temp_test = 6
print (test_data[temp_test][test_label[temp_test]],test_label[temp_test])
autoencoder
def frange(x, y, jump):
  while x < y:
    yield x
    x += jump
```

```
EbNodB_range = list(frange(-4,8.5,0.5))
ber = [None]*len(EbNodB_range)
for n in range(0,len(EbNodB_range)):
    EbNo=10.0**(EbNodB_range[n]/10.0)
    noise_std = np.sqrt(1/(2*R*EbNo))
    noise_mean = 0
    no_errors = 0
    nn = N
    noise = noise_std * np.random.randn(nn,n_channel)
    encoded_signal = encoder.predict(test_data)
    final_signal = encoded_signal + noise
    pred_final_signal =  decoder.predict(final_signal)
    pred_output = np.argmax(pred_final_signal,axis=1)
    no_errors = (pred_output != test_label)
    no_errors =  no_errors.astype(int).sum()
    ber[n] = no_errors / nn
    print ('SNR:',EbNodB_range[n],'BER:',ber[n])

import matplotlib.pyplot as plt
plt.plot(EbNodB_range, ber, 'bo',label='Autoencoder(7,4)')
#plt.plot(list(EbNodB_range), ber_theory, 'ro-',label='BPSK BER')
plt.yscale('log')
plt.xlabel('SNR Range')
plt.ylabel('Block Error Rate')
plt.grid()
plt.legend(loc='upper right',ncol = 1)
plt.savefig('AutoEncoder_7_4_BER_matplotlib')
plt.show()
x_emb = encoder.predict(test_data)
noise_std = np.sqrt(1/(2*R*EbNo_train))
noise = noise_std * np.random.randn(N,n_channel)
x_emb = x_emb + noise
from sklearn.manifold import TSNE
X_embedded = TSNE(learning_rate=700, n_components=2,n_iter=35000, random_state=0,
perplexity=60).fit_transform(x_emb)
print (X_embedded.shape)
X_embedded = X_embedded / 7
import matplotlib.pyplot as plt
plt.scatter(X_embedded[:,0],X_embedded[:,1])
#plt.axis((-2.5,2.5,-2.5,2.5))
plt.grid()
plt.show()
```