基于深度学习信道译码的改进

王振伟

摘要

复现论文中[1]讨论了使用深度神经网络对随机码和结构码如极化码进行一次性译码的可能性，作者发现这种方法对于两种码字在较短的长度下都可以达到最大后验概率译码方法的性能。但是由于训练神经网络时所需的数据集大小随信息比特位位数 K 的增长呈指数增长，因此该方法最多只能译码 K=24,N=64（N 为码长）的极化码。对于神经网络译码而言，生成 2 的 64 次方规模的训练集并且学习如何解码无论从时间或是空间的角度上来说都是不可能实现的。我们希望利用极化码的结构特性将 K=64,N=128 的极化码拆分成若干较小的、可以使用神经网络译码的部分分别译码，从而译码 K=64,N=128 的极化码，拓展神经网络译码极化码的译码长度。

关键词：极化码；神经网络；编码结构；译码长度

1 引言

传统的极化码译码方法如 SC、SC-list 都是顺序译码方案，即逐个译码，虽然误码率等性能很好，但是译码速度却受限于该特性，同时对硬件设计也不友好。复现论文中提出的运用神经网络译码极化码的方法虽然目前最多只能译码 K=24,N=64 的极化码，但是它一次性译码的特性使得该方法在译码速度上颇具潜力。如果能解决其只能处理短码的问题，则该方法有望成为极化码主流译码算法。因此尝试拓展该方法的译码长度是有价值的工作。

2 相关工作

90 年代出现了关于使用神经网络进行译码的不同想法。虽然在[2]中，输出节点表示码字的位，但也可以对每个码字[3]使用一个输出节点。对于汉明编码，另一种变体是只使用特征作为神经网络的输入，以找到最有可能的错误模式[4]。随后，卷积码的 NND 出现在 1996 年，当时 Wang 和 Wicker 证明了 NND 与理想的维特比解码器[5]的性能相匹配。但他们也提到了 NND 的一个非常重要的缺点：译码问题比传统的模式识别问题有更多的可能性。这就限制了 NND，它只能译码短码。然而，利用递归神经网络[6]进一步改进了卷积码的神经网络译码器。NND 在定长码和卷积码方面都没有取得任何大的突破。由于当时的标准训练技术，这两种码不可能与使用大量神经元和层的神经网络一起工作，这也是人们认为神经网络译码不适合译码较长的码字的原因。因此，人们对神经网络的兴趣减少了，不仅是对机器学习本身，而且将其应用在译码方面的兴趣也减少了。在接下来的几年里，有了一些轻微的改进，例如，通过使用随机神经网络[7]或通过减少权重[8]的数量。2006 年，一种新的训练技术，即逐层无监督预训练，然后是梯度下降微调[9]，导致了神经网络的复兴，因为它能训练具有更多层的神经网络。具有许多隐藏层的神经网络被称为深度神经网络。如今，强大的新硬件，如图形处理单元（gpu）可以用来加速学习和推理。在神经网络的复兴中，新的 NND 思想出现了。然而，与以前的工作相比，神经网络学习技术仅用于优化已知的解码方案，我们称之为专家知识的引入。例如，在[10]中，给定置

信度传播（BP）算法的 Tanner 图权重，并通过神经网络技术进行学习，以改进 BP 算法。机器学习的最新进展似乎还没有适应于学习译码的领域。

3 本文方法

3.1 本文方法概述

本报告所做的工作重点在于改进该方法的性能，扩展神经网络的译码长度，在此先对复现论文中使用神经网络进行极化码译码工作的方法进行概述：在给定的 K、N 的条件下生成所有可能的极化码码字作为训练集，神经网络中的噪声层会模拟 BPSK 调制、添加高斯白噪声的过程，之后的译码层会学习如何译码这些添加了噪声的码字。

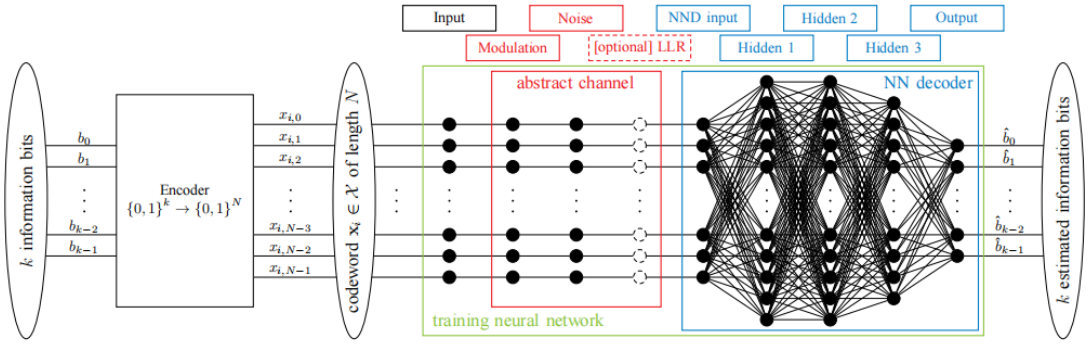

图 1: 复现论文方法示意图

现将改进的工作概述如下：由极化码的编码结构可知，我们可以将 K=64、N=128 的极化码译码工作分解为 K=15、N=64,K=18、N=32,K=31、N=32 的三部分进行译码，这中间的变换只需要一些数学计算即可。前两部分的工作我们可以用两个相应的神经网络的译码层完成，而第三部分的译码工作则因为此部分的码字基本不具备纠错能力，没有使用神经网络译码的必要，同时该部分在整段码字的最下端，本身极化结构未受到其它部分影响，因此直接将接收的 Y 硬判决为 0、1，直接计算即可译码。

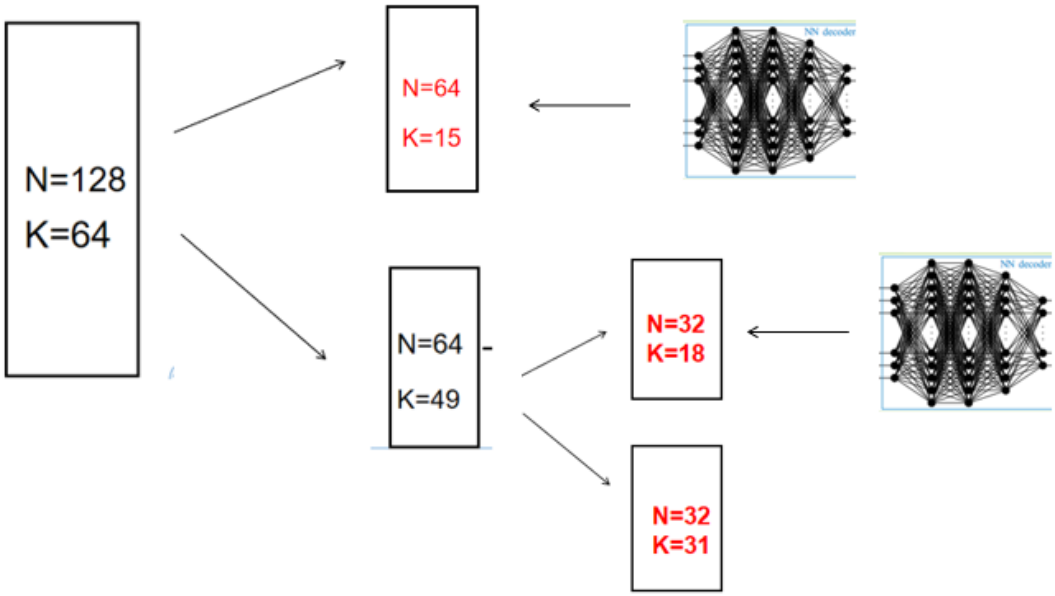

图 2: 改进方法示意图

3.2 不同模块之间的转换

参考极化码译码方法中经典的 SC(串行抵消算法) 译码算法即可知我们只需对收到的、模拟添加高斯白噪声的 Y 序列这 128 个 bit 进行 64 次 f 函数的运算即可得到 K=15、N=64 的神经网络解码层的输入 LLR。将 K=15、N=64 模块的译码结果进行 N=64 的极化码编码之后作为 g 函数的参数可得到下半部分 K=49、N=64 的码字的 LLR 的输入，同理可得到 K=18、N=32 模块的输入。而最后一部分 K=31、N=32 的运算则只需取收到的 Y 序列的后 32 个 bit, 将其硬判决为 0、1，由于该段第 1 个 bit 为冻结比特，始终为 0，因此若第 1 个 bit 不为 0, 将其翻转再译码。下面（1）为 f 函数，（2）为 g 函数，sign 为取符号函数。

$$LLR1 = sign(L1)sign(L2)min|L1|,|L2| \tag{1}$$

$$LLR2 = (1-2U1)L1 + L2 \tag{2}$$

4 复现细节

4.1 与已有开源代码对比

本报告中的工作建立在已有复现论文代码的基础上完成，重点在于对复现论文工作的改进（创新增量）。因此下面展示改进工作所添加的伪代码

Procedure 1 Decoding K=64, N=128 noisy polarization code

Input: K=64, N=128 noisy polarization code *Y,* decoder *K15N64,* decoder *K18N32*
Output: Ber（误码率）
for *l* ***in*** *Y64*
do
v_l = f(Y_l, Y_{l+64})　　u_l = decoderK15N64(v_l)
end
for *i* ***in*** *U64*
do
T_i = Encoder(U)　　P = g($Y_{64,127}$, Ti)　　$U_{i,i+64}$ = decoderK18N32(P)

$U_{97,127}$ = decoder($Y_{97,127}$)
end

本报告中的工作中我首先使用了 K=1,N=8,K=7,N=8 的两个训练好的神经网络尝试对 K=8,N=16 的有噪声的极化码进行解码。效果如下图

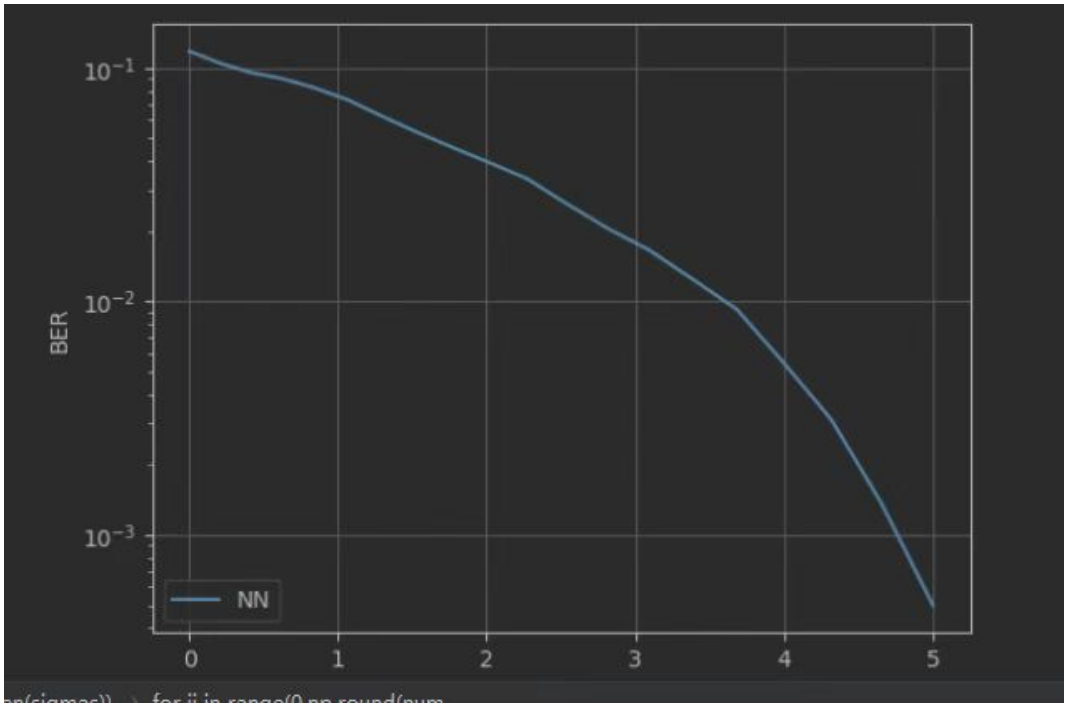

图 3: K=8,N=16 译码结果

之后在使用作者给定参数训练 K=15,N=64 和 K=18,N=32 的两个模型时，发现误码率不佳。

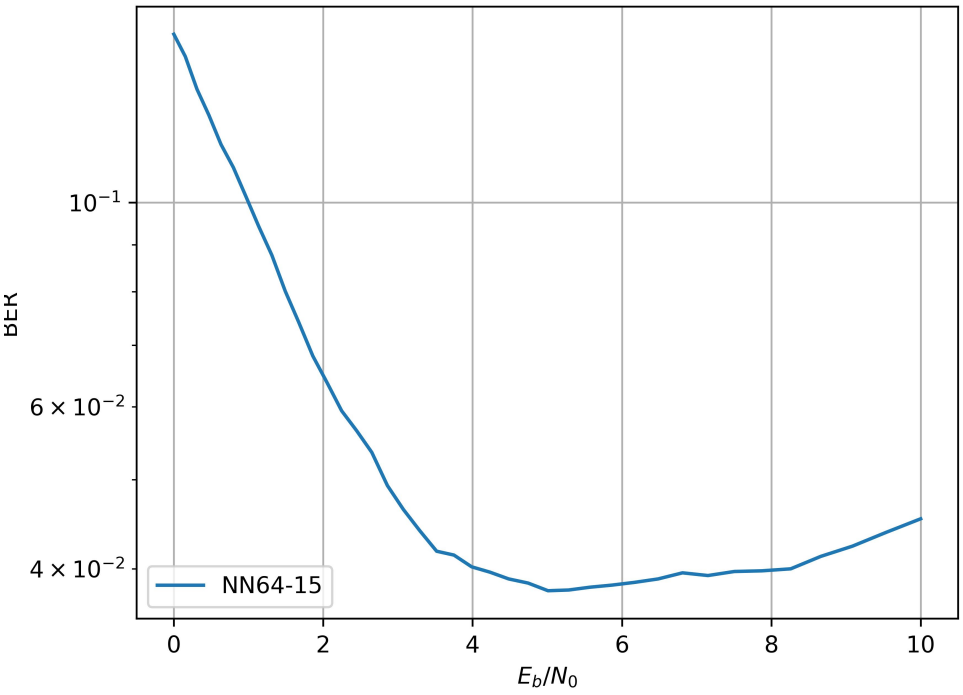

图 4: K=15,N=64

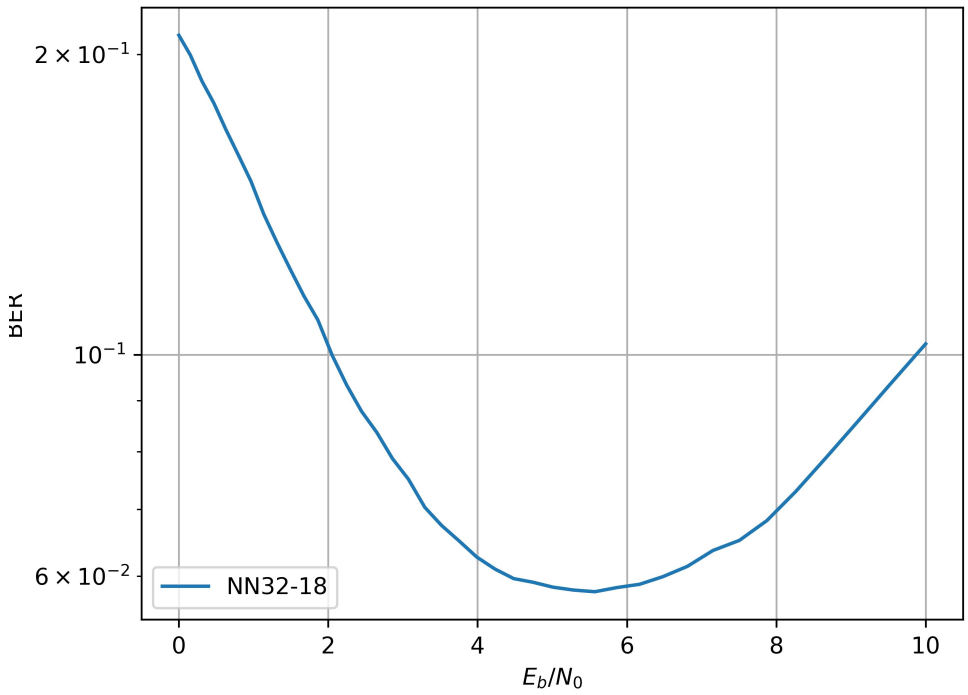

图 5: K=18,N=32

在查阅资料修改训练参数之后训练了 30 余个模型之后，我们选取了两个较好的作为译码 K=64,N=128 的模型。

my_decoder_k14N16_5.h5	2 664	202
my_decoder_k15N64.h5	2 857	202
my_decoder_k15N64_1.h5	2 857	202
my_decoder_k15N64_2048.h5	10 815	202
my_decoder_k15N64_20480....	10 815	202
my_decoder_k15N64_5.h5	2 857	202
my_decoder_k16N16_5.h5	2 666	202
my_decoder_k17N32.h5	2 731	202
my_decoder_k17N32_1.h5	2 731	202
my_decoder_k17N64.h5	2 859	202
my_decoder_k18N32.h5	2 732	202
my_decoder_k18N32_10242.h5	2 732	202
my_decoder_k18N32_12.h5	2 732	202
my_decoder_k18N32_20481....	10 565	202
my_decoder_k18N32_3.5.h5	2 732	202
my_decoder_k18N32_3.5_2.h5	2 658	202
my_decoder_k18N32_4.h5	2 732	202
my_decoder_k18N32_5.h5	2 732	202
my_decoder_k18N32_6.5.h5	2 732	202
my_decoder_k18N32_6_2.h5	10 565	202
my_decoder_k18N32_7.h5	2 732	202
my_decoder_k18N32_8.h5	2 732	202

图 6: 训练若干模型

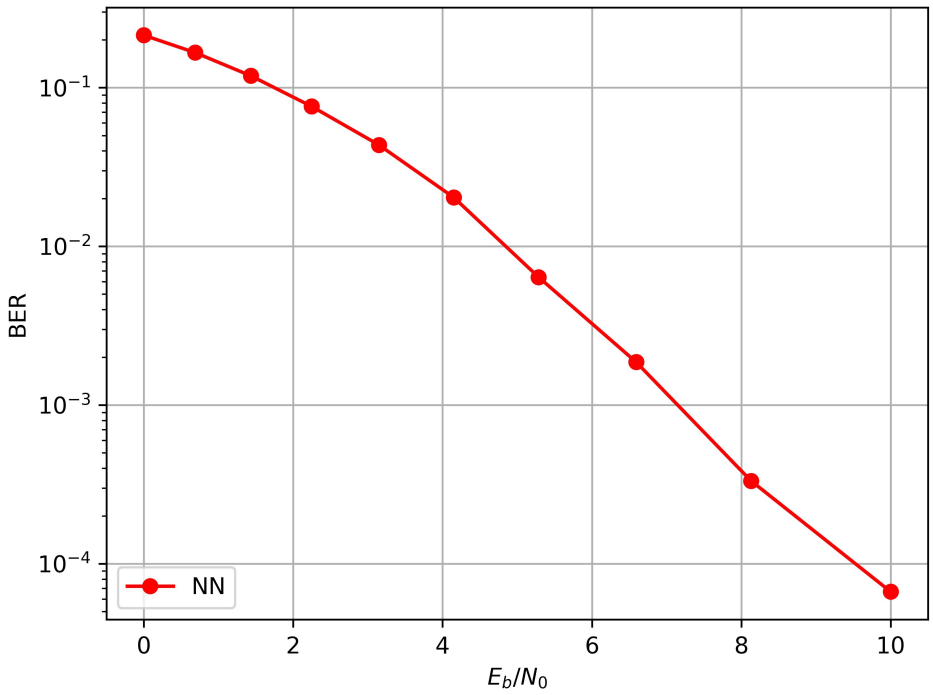

图 7: 优化后的 K15N64

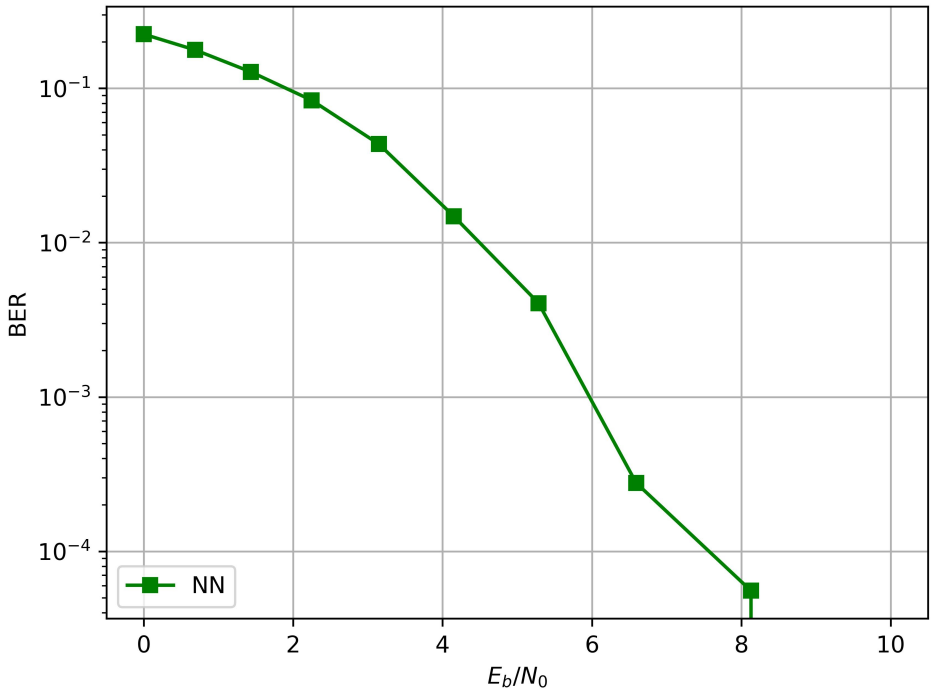

图 8: 优化后的 K18N32

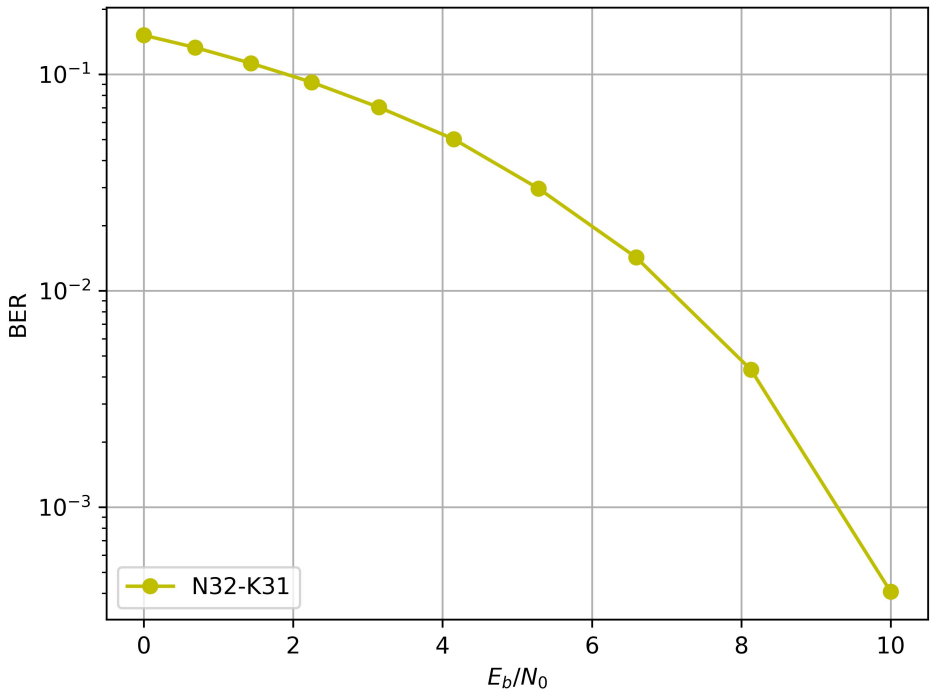

图 9: K31N32

4.2 实验环境搭建

本报告复现代码需在服务器上 linux 系统下配置 tenserflow 环境，导入 Keras,matplotlib,numpy 库方可运行。

4.3 界面分析与使用说明

复现报告代码中包括两个 py 文件，若需要训练并生成不同模型，请调整参数运行 decoderPro.py 文件。若需要测试译码 K=64,N=128 的极化码，请运行 decodeK64N128.py 文件（确保 decoderk15N64.h5、decoderK18N32.h5 文件在同一文件夹）

4.4 创新点

本报告的创新点即是扩展了神经网络的译码长度，使原本最多只能译码 K=24，N=64 长度的极化码的神经网络能译码 K=64,N=128 的极化码。

5 实验结果分析

本实验结果证明了通过拆分 K=64,N=128 的极化码的方式能够实现神经网络译码长度的拓展。同时下图的误码率曲线可知该方法的译码效果也是较好的。

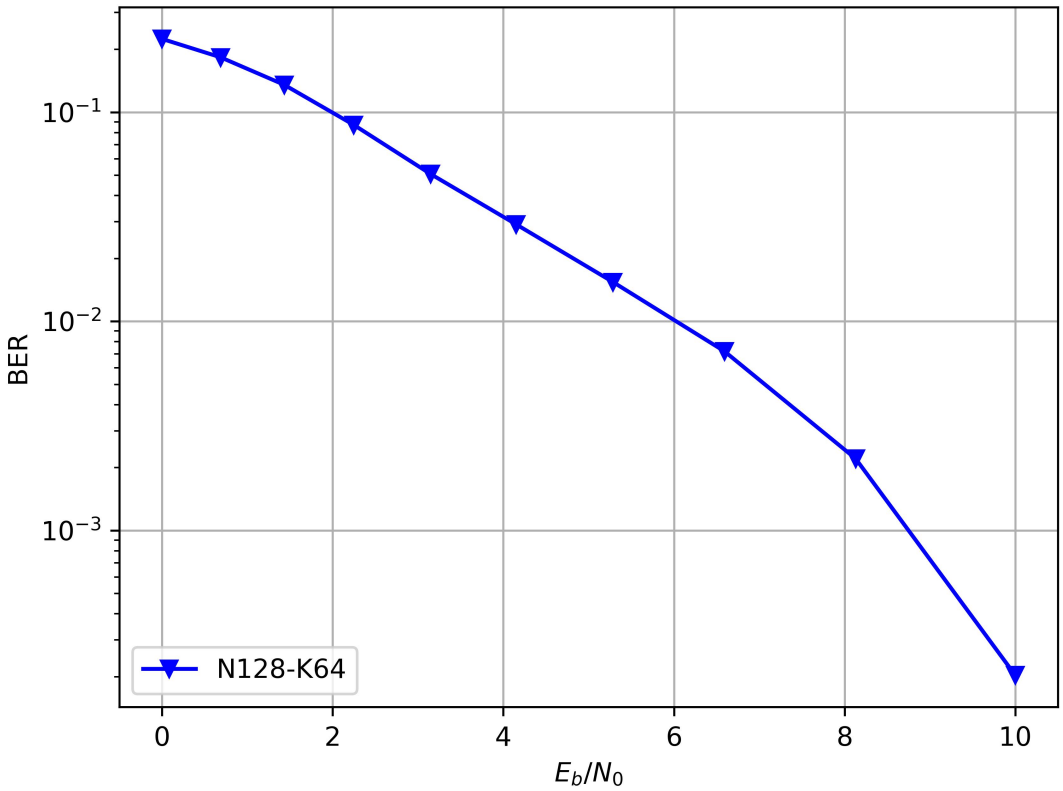

图 10: 实验结果示意: 译码 K=64,N=128 的极化码

6 总结与展望

在本报告中，我对神经网络译码极化码的译码能力进行了拓展，使得神经网络对极化码的译码能力从 K=24，N=64 提升到 K=64,N=128, 突破了神经网络译码长度因训练数据集过大受限的译码长度。但是该方法存在的不足在于拆分后有一些部分的码率很高（K/N）, 该部分码字单独来看不具有纠错能力，只能直接译码，但是对于极化码整体来看恰恰是这部分占据了最好的信道。这就相当于浪费了最好的这部分信道。本复现论文中作者让神经网络学习的译码方法为典型的 SC 译码（一种顺序译码方法），而在将来可能的方向是让神经网络学习 BP 译码（置信度传播译码方法）这种并行译码算法，这也许能使神经网络的译码能力有所提升。

参考文献

[1] GRUBER T, CAMMERER S, HOYDIS J, et al. On Deep Learning-Based Channel Decoding[C]//IEEE. 2017.

[2] CAID W R, MEANS R W. Neural network error correcting decoders for block and convolutional codes [C]//IEEE Global Telecommunications Conference. 1990.

[3] STEFANO A D, MIRABELLA O, CATALDO G D, et al. On the use of neural networks for Hamming coding[C]//Circuits and Systems, 1991., IEEE International Sympoisum on. 2002.

[4] TALLINI L G, CULL P. Neural nets for decoding error-correcting codes[C]//IEEE Technical Applications Conference and Workshops. Northcon/95. Conference Record.

[5] WANG X A, WICKER S B. An artificial neural net Viterbi decoder[J]. IEEE Trans Communications, 1996, 44(2): 165-171.

[6] HAMALAINEN A, HENRIKSSON J. A recurrent neural decoder for convolutional codes[C]//IEEE International Conference on Communications. 1999.

[7] ABDELBAKI H, GELENBE E, EL-KHAMY S E. Random neural network decoder for error correcting codes[C]//International Joint Conference on Neural Networks. 1999: 3241-3245.

[8] JA-LING W U, TSENG Y H, HUANG Y M. NEURAL NETWORK DECODERS FOR LINEAR BLOCK CODES[J]. International Journal of Computational Engineering Science, 2002.

[9] HINTON G E, OSINDERO S, TEH Y W. A fast learning algorithm for deep belief nets[J]. MIT Press, 2006(7).

[10] NACHMANI E, BE'ERY Y, BURSHTEIN D. Learning to decode linear codes using deep learning [C]//2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton). 2016.