CHANNEL DECODING VIA MACHINE LEARNING

BY

SHIBARA ASHWIN HEBBAR

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Adviser:

Professor Pramod Viswanath

ABSTRACT

Error-correcting codes (codes) are the backbone of the modern information age and were essential to the invention of groundbreaking technology such as WiFi, cellular, cable, and satellite modems. In this thesis, we focus on using machine learning techniques to design efficient and reliable data-driven decoders for state-of-the-art channel codes.

In the first half of the thesis, we introduce a neural-augmented decoder for Turbo codes called TinyTurbo. TinyTurbo has complexity comparable to the classical max-log-MAP algorithm but has much better reliability than the max-log-MAP baseline and performs close to the MAP algorithm. We show that TinyTurbo exhibits strong robustness on a variety of practical channels of interest, such as EPA and EVA channels, which are included in the LTE standards. We also show that TinyTurbo strongly generalizes across different rate, blocklengths, and trellises.

In the second half of the thesis, we focus on designing data-driven decoders for the polar code family: Polar codes and PAC codes. We pose the decoding of PAC codes as a tree-search problem, and introduce PAC-DQN, a reinforcement learning based decoder. While PAC-DQN achieves a near-optimal reliability for short codes, it suffers from poor training sample complexity and is not scalable to larger codes. We then introduce CRISP, a GRU-powered neural decoder, which uses curriculum learning to achieve excellent reliability and scale to larger codes. We show that CRISP outperforms the successive-cancellation (SC) decoder and attains near-optimal reliability performance on the Polar$(16, 32)$, Polar$(22, 64)$ and PAC$(16, 32)$ codes.

To my parents and brother, for their love and support.

ACKNOWLEDGMENTS

I would first like to thank my advisor Prof. Pramod Viswanath for introducing me to this problem, and for his constant guidance and support. I convey my sincere gratitude to my collaborators Prof. Sewoong Oh and Prof. Hyeji Kim for their insightful guidance.

I am extremely grateful to Ashok Makkuva for being an excellent mentor. I also thank my collaborators Prof. Suma Bhat, Viraj Nadkarni, Rajesh Mishra, Sravan Kumar Ankireddy, Xiyang Liu, and Dr. Yihan Jiang.
I thank all my friends who have supported me along the way.

I am deeply indebted to my brother Achinthya, parents, and family for their unwavering support and encouragement throughout my studies. I dedicate this thesis to them.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Shannon, in his groundbreaking work in 1948 [1], showed that information can be reliably transmitted over a noisy channel if the capacity of the transmission is less than the channel capacity. One of the fundamental methods that render this possible is channel coding. Codes, composed of (encoder, decoder) pairs, ensure reliable data transmission even under noisy conditions. The encoder adds structured redundancy to the data, enabling the decoder to perform error correction and detection. Since Shannon's work, several landmark codes have been proposed: Convolutional codes, low-density parity-check (LDPC) codes, Turbo codes, Polar codes, and more recently, Polarization-Adjusted-Convolutional (PAC) codes [2].

Turbo codes [3] have been widely used in modern communication systems and are part of 3G and 4G standards. While Turbo codes operate at near-optimal performance on the canonical additive white Gaussian noise (AWGN) channel, it is well known that the classical Turbo decoder lacks robustness and performs poorly on non-AWGN channels. The MAP iterative Turbo decoder is computationally expensive; therefore, approximations such as the max-log-MAP decoder are used in practice, trading reliability for complexity.

Polar codes, introduced by Arikan [4], are widely used in practice owing to their reliable performance in the short blocklength regime. They exhibit several crucial information-theoretic properties; practical finite-length performance, however, depends on high complexity decoders. This search for the design of efficient and reliable decoders for polar codes has been the focus of substantial research in the past decade.

PAC codes [5], variants of polar codes, further improve performance, nearly achieving the fundamental lower bound on the performance of any code at finite lengths, albeit at a higher decoding complexity [6]. The sequential "Fano decoder" [7] allows PAC codes to perform information-theoretically

near-optimally; however, the decoding time is long and variable [8].

Thus, designing decoders with high reliability and robustness is of great practical interest. This thesis focuses on designing efficient and reliable decoders for Turbo, Polar, and PAC codes.

Deep learning has been highly successful in recent years across various disciplines such as computer vision and natural language processing. Deep learning for communication [9, 10] has been an active field in the recent years and has seen success in many problems including symbol detection [11, 12, 13, 14, 15, 16], channel estimation [17, 18, 19], and channel feedback [20, 21], among others.

We have also seen impressive results from deep-learning-based decoders in channel decoding [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38]. There have also been works that focus on jointly learning channel encoder-decoder pairs [39, 23, 40, 41, 42, 43, 44, 45, 46, 47]. It is well known in the literature that naively parameterizing the decoder by general-purpose neural networks does not work well, and they perform poorly even for small blocklengths like $n = 16$ [48]. Hence it is essential to use efficient decoding architectures that capitalize on the structure of the encoder [37, 27]. These results demonstrate the potential of DL in designing optimal decoders; however most of them still suffer from substantial computational complexity, making them intractable to be deployed in real-world communication systems.

Model-based machine learning is an alternate approach that has gained traction in recent years [9]. One of the key ideas in this approach is to augment learnable parameters to existing channel decoders and train these parameters. A strong benefit of such neural decoders is that the complexity of the decoder does not increase much while reliability can be improved.

The rest of this thesis is organized as follows: Chapter 2 formally defines the channel decoding problem and introduces the codes of interest. Chapter 3 proposes a model-based machine learning approach to decode Turbo codes. Chapter 4 proposes a reinforcement-learning based approach to decode PAC codes. An improved approach is introduced in Chapter 5, which proposes a sequential neural decoder, trained via supervised learning, that can be trained to decode Polar codes and PAC codes.

CHAPTER 2

BACKGROUND

In this chapter, we formally define the channel decoding problem and provide a background of Turbo, Polar, and PAC codes. We also introduce some of the classical decoding algorithms for these codes.

Our notation is the following: we denote Euclidean vectors by small bold face letters $\boldsymbol{x}, \boldsymbol{y}$, etc. $[n] \triangleq \{1, 2, \ldots, n\}$. For $\boldsymbol{m} \in \mathbb{R}^n, \boldsymbol{m}_{<i} \triangleq (m_1, \ldots, m_{i-1})$. $\mathcal{N}(0, \boldsymbol{I}_n)$ denotes a standard Gaussian distribution in $\mathbb{R}^n$. $\boldsymbol{u} \oplus \boldsymbol{v}$ denotes the bitwise XOR of two binary vectors $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^\ell$.

2.1 Channel decoding

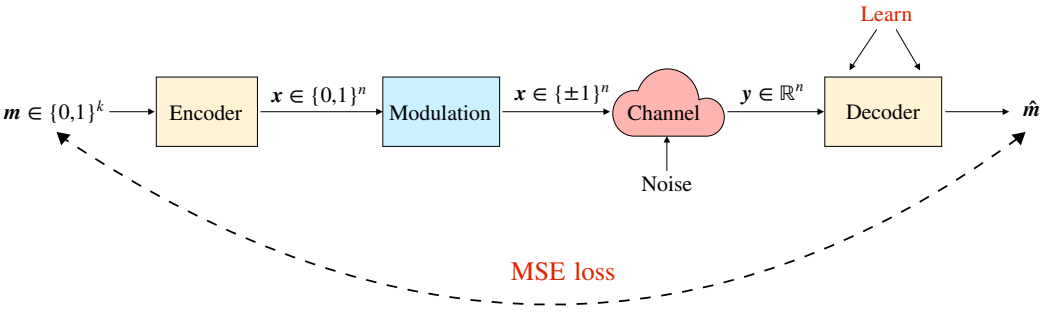


Figure 2.1: Channel decoding problem.

The primary goal of channel decoding is to design efficient decoders that can correctly recover the message bits upon receiving codewords corrupted by noise (Figure 2.1). More precisely, let $\boldsymbol{u} = (u_1, \ldots, u_k) \in \{0,1\}^k$ denote a block of *information/message* bits that we wish to transmit. An encoder $g : \{0,1\}^k \to \{0,1\}^n$ maps these message bits into a binary codeword $\boldsymbol{x}$ of length n, i.e. $\boldsymbol{x} = g(\boldsymbol{u})$. The *rate* of a code is defined as $R = \frac{k}{n}$, the ratio of the number of message bits and the codeword length. The code rate represents the amount of redundancy in the code. The encoded bits $\boldsymbol{x}$ are

modulated via Binary Phase Shift Keying (BPSK), i.e. $\boldsymbol{x} \mapsto 1-2\boldsymbol{x} \in \{\pm 1\}^n$, and are transmitted across the channel. We denote both the modulated and unmodulated codewords as $\boldsymbol{x}$. The channel, denoted as $P_{Y|X}(\cdot|\cdot)$, corrupts the codeword $\boldsymbol{x}$ to its noisy version $\boldsymbol{y} \in \mathbb{R}^n$. Upon receiving the corrupted codeword, the decoder f_θ estimates the message bits as $\hat{\boldsymbol{u}} = f_\theta(\boldsymbol{y})$. The performance of the decoder is measured using standard error metrics such as Bit-Error-Rate (BER) or Block-Error-Rate (BLER):

$$\mathrm{BER}(f_\theta) \triangleq (1/k) \sum_i \mathbb{P}[\hat{\boldsymbol{u}}_i \neq \boldsymbol{u}_i]$$

$$\mathrm{BLER}(f_\theta) \triangleq \mathbb{P}[\hat{\boldsymbol{u}} \neq \boldsymbol{u}]$$

Given an encoder g with code parameters (k, n) and a channel $P_{Y|X}$, the channel decoding problem can be mathematically formulated as:

$$\theta \in \arg\min_\theta \mathrm{BER}(f_\theta), \tag{2.1}$$

which is a joint classification of k binary classes. To train the parameters θ, we use the mean-square-error (MSE) or binary cross-entropy (BCE) loss as a differentiable surrogate to the objective in (2.1).

2.2 Turbo codes

2.2.1 Turbo encoder

Turbo encoder consists of an interleaver and two identical Recursive Systematic Convolutional (RSC) encoders, denoted by π and (E_1, E_2) respectively, as depicted in Figure 2.2. RSC codes are a type of convolutional code obtained by feeding back one of the outputs of a conventional convolutional encoder to its input [49]. Let $\boldsymbol{u} = (u_1, \ldots, u_K)$ denote the sequence of K information bits that we wish to transmit. To encode the message bits $\boldsymbol{u}$, a block of them is directly transmitted via the systematic bit sequence $\boldsymbol{x}^s = (x_1^s, x_2^s, \ldots, x_K^s)$. On the other hand, the encoder E_1 generates the parity bit sequence $\boldsymbol{x}^{1p} = (x_1^{1p}, x_2^{1p}, \ldots, x_k^{1p})$ from $\boldsymbol{u}$ whereas the encoder E_2 generates the parity sequence $\boldsymbol{x}^{2p} = (x_1^{2p}, x_2^{2p}, \ldots, x_k^{2p})$ using the interleaved

input $\widetilde{\boldsymbol{u}} = \pi(\boldsymbol{u})$. We assume that the encoded bits $\boldsymbol{x} = (\boldsymbol{x}^s, \boldsymbol{x}^{1p}, \boldsymbol{x}^{2p})$ are modulated via Binary Phase Shift Keying (BPSK) and transmitted over the channel. For concreteness, we consider LTE Turbo codes, for which the RSC code has the generator matrix $(1, g_1(D)/g_2(D))$ with $g_1(D) = 1 + D^2 + D^3$ and $g_2(D) = 1 + D + D^3$ [50].

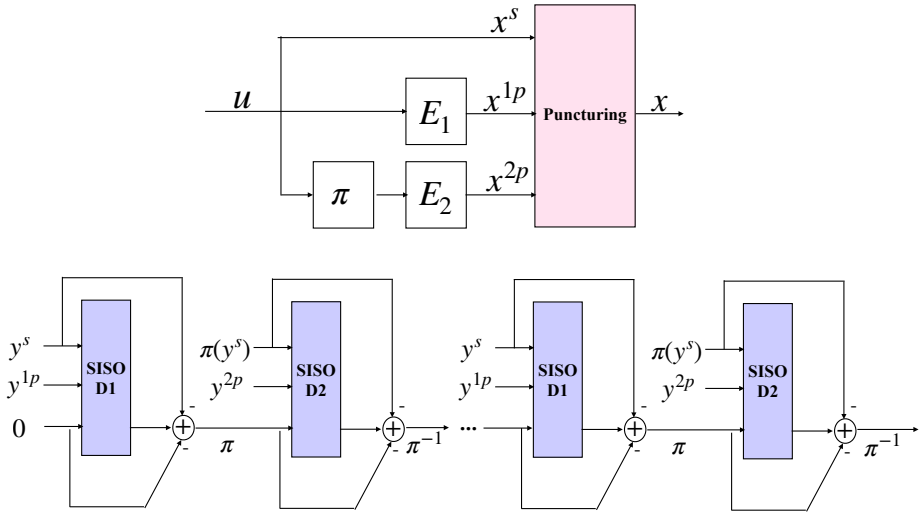


Figure 2.2: Turbo encoder (top); Turbo decoder (bottom)

2.2.2 Turbo decoder

As depicted in Figure 2.2 (bottom), the classical turbo decoder [3] contains two identical Soft-Input Soft-Output (SISO) decoders (D_1, D_2) and relies on the 'Turbo principle', where these SISO decoders iteratively refine the posterior of the information bits. More precisely, each SISO decoding block takes the received signals and the prior as input and estimates the posterior distribution for the message bits. This is then fed as a prior for the next SISO decoding block. This procedure is repeated for a fixed number of iterations to estimate the final posterior. The celebrated BCJR algorithm [51] is used as the SISO decoding block; this algorithm performs a maximum a-posteriori (MAP) decoding and is the optimal decoder for convolutional codes. We describe the BCJR decoding in the context of decoder D_1:

MAP turbo decoder: Let $\boldsymbol{y} = (\boldsymbol{y}^s, \boldsymbol{y}^{1p})$ be the log-likelihood-ratios (LLRs) of the encoded bits corresponding to encoder E_1. The BCJR SISO decoder D_1 takes the soft-information $\boldsymbol{y}$ as its input and obtains the LLRs for the

information bits as follows:

$$L(u_k|\boldsymbol{y}) \triangleq \log \frac{\mathbb{P}\left(u_k=1|\boldsymbol{y}\right)}{\mathbb{P}\left(u_k=0|\boldsymbol{y}\right)} = \log \frac{\sum_{(s',s)\in S^1} \mathbb{P}\left(s',s,\boldsymbol{y}\right)}{\sum_{(s',s)\in S^0} \mathbb{P}\left(s',s,\boldsymbol{y}\right)},$$

where $S^1 = \{(s',s) : u_k = 1\}$ denotes the set of all ordered pairs (s',s) corresponding to state transitions $s' \to s$ caused by data input $u_k = 1$, whereas $S^0 = \{(s',s) : u_k = 0\}$ pertains to the input $u_k = 0$. Assuming that the underlying channel is AWGN and memoryless, the joint probabilities $\mathbb{P}\left(s',s,\boldsymbol{y}\right)$ can be efficiently computed:

$$\begin{aligned}\mathbb{P}\left(s',s,\boldsymbol{y}\right) &= \mathbb{P}\left(s',s,\boldsymbol{y}_1^{k-1},y_k,\boldsymbol{y}_{k+1}^{K}\right)\\ &= \mathbb{P}\left(s',\boldsymbol{y}_1^{k-1}\right)\mathbb{P}\left(y_k,s|s'\right)\mathbb{P}\left(\boldsymbol{y}_{k+1}^{K}|s\right)\\ &= \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s),\end{aligned}$$

where $\alpha_{k-1}(s') \triangleq \mathbb{P}\left(s',\boldsymbol{y}_1^{k-1}\right)$ and $\beta_k(s) \triangleq \mathbb{P}\left(\boldsymbol{y}_{k+1}^{K}|s\right)$ can be computed via the forward and backward recursions [51]:

$$\begin{aligned}\alpha_k(s) &= \sum_{s'\in S_R} \alpha_{k-1}(s')\gamma_k(s',s),\\ \beta_{k-1}(s') &= \sum_{s\in S_R} \beta_k(s)\gamma_k(s',s)\end{aligned}$$

with the initial conditions $\alpha_0(s) = \beta_K(s) = \mathbb{1}\{s=0\}$. Here $S_R = \{0,1,\ldots,2^m-1\}$ denotes the set of all possible states for the encoder whose memory is m. The branch transition probabilities $\gamma_k(s',s) \triangleq \mathbb{P}\left(y_k,s|s'\right)$ can be computed as

$$\gamma_k(s',s) = \exp\left(\frac{1}{2}\left(x_k^s y_k^s + x_k^{1p} y_k^{1p}\right) + \frac{1}{2}u_k L(u_k)\right), \tag{2.2}$$

where $L(u_k)$ is the *apriori* LLR for the bit u_k. To ensure numerical stability, we consider the logarithmic values of the above probabilities, i.e. $\bar{\alpha}_k(s) \triangleq$

$\log \alpha_k(s), \bar{\beta}_k(s) \triangleq \log \beta_k(s)$, and $\bar{\gamma}_k(s', s) \triangleq \log \gamma_k(s', s)$, and obtain

$$\begin{aligned}
\bar{\gamma}_k(s', s) &= \frac{1}{2}\left(x_k^s y_k^s + x_k^{1p} y_k^{1p}\right) + \frac{1}{2} u_k L(u_k), \\
\bar{\alpha}_k(s) &= \mathrm{LSE}_{s' \in S_R}\left(\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k(s', s)\right), \\
\bar{\beta}_{k-1}(s') &= \mathrm{LSE}_{s \in S_R}\left(\bar{\beta}_k(s) + \bar{\gamma}_k(s', s)\right), \\
L(u_k|\boldsymbol{y}) &= \mathrm{LSE}_{(s',s)\in S^1}\left(\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k(s', s) + \bar{\beta}_k(s)\right) \\
&\quad - \mathrm{LSE}_{(s',s)\in S^0}\left(\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k(s', s) + \bar{\beta}_k(s)\right)
\end{aligned} \tag{2.3}$$

where $\mathrm{LSE}(z_1, \ldots, z_n) \triangleq \log(\exp(z_1) + \ldots + \exp(z_n))$, the log sum-of-exponentials function. Upon obtaining the posterior LLR $L(u_k|\boldsymbol{y})$, the decoder D_1 computes the extrinsic LLR $L_e(u_k)$ as

$$L_e(u_k) = L(u_k|\boldsymbol{y}) - L(y_k^s) - L(u_k), \quad k \in [K] \tag{2.4}$$

which is interleaved and passed as a prior to the decoder D_2. This message passing decoding is iteratively performed M times. The posterior $L^M(u_k|\boldsymbol{y})$ after the M^{th} iteration is used to estimate the message bits:

$$\hat{u_k} = \mathbb{1}\{L^M(u_k|\boldsymbol{y}) < 0\}$$

2.2.3 Max-log-MAP turbo decoder

The *MAP* algorithm uses the exact LSE function in the above set of equations in Eq. (2.3) which is computationally expensive. Hence, in practice, several variants and approximations are often used. A popular such decoder is the *max-log-MAP* algorithm. The main idea behind the max-log-MAP is to approximate the LSE function by the maximum, i.e.

$$\mathrm{LSE}(z_1, \ldots, z_n) \approx \max(z_1, \ldots, z_n), \quad z_1, \ldots, z_n \in \mathbb{R}.$$

While the max-log-MAP algorithm is more efficient than the MAP, it is less reliable than the MAP [52].

2.3 Polar codes

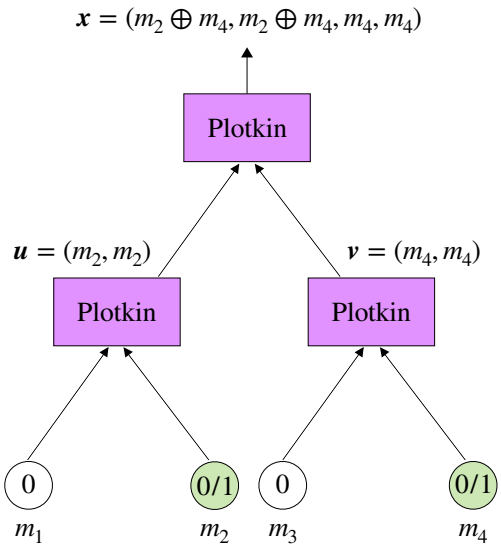


Figure 2.3: Polar$(2, 4)$: Polar encoding via Plotkin tree

2.3.1 Polar encoding

Polar codes have a recursive encoding method, defined as follows: let (k, n) be the code parameters with $n = 2^p, 1 \leq k \leq n$. In order to encode a block of message bits $\boldsymbol{u} = (u_1, \ldots, u_k) \in \{0,1\}^k$, we first embed them into a source message vector $\boldsymbol{m} \triangleq (m_1, \ldots, m_n) = (0, \ldots, u_1, 0, \ldots, u_2, 0, \ldots, u_k, 0, \ldots) \in \{0,1\}^n$, where $\boldsymbol{m}_{I_k} = \boldsymbol{u}$ and $\boldsymbol{m}_{I_k^C} = 0$ for some $I_k \subseteq [n]$. Since the message block $\boldsymbol{m}$ contains the information bits $\boldsymbol{u}$ only at the indices pertaining to I_k, the set I_k is called the *information set*, and its complement I_k^C the *frozen set*. For the set I_k, we first compute the capacities of the n individual polar bit channels and rank them in their increasing order [53]. Then I_k picks the top k out of them. For example, Polar$(2, 4)$ has the ordering $m_1 < m_2 = m_3 < m_4$ and $I_k = \{2, 4\}$, and thus $\boldsymbol{m} = (0, m_2, 0, m_4)$. Similarly, Polar$(4, 8)$ has $m_1 < m_2 < m_3 < m_5 < m_4 < m_6 < m_7 < m_8$, $I_4 = \{4, 6, 7, 8\}$ and $\boldsymbol{m} = (0, 0, 0, m_4, 0, m_6, m_7, m_8)$.

Finally, we obtain the polar codeword $\boldsymbol{x} = \text{PlotkinTree}(\boldsymbol{m})$, where the mapping PlotkinTree : $\{0,1\}^n \to \{0,1\}^n$ is given by a complete binary tree,

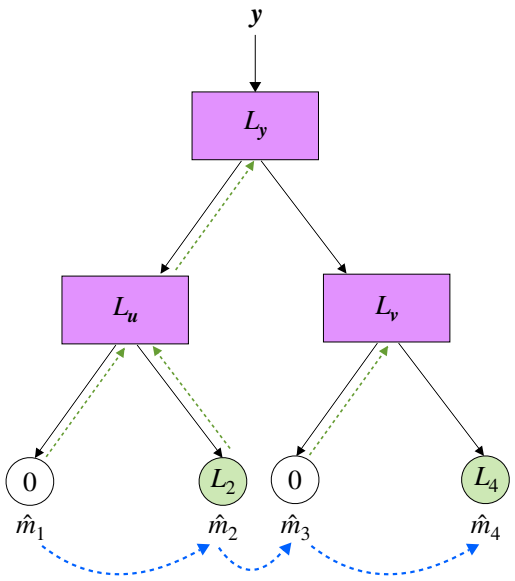


Figure 2.4: Polar$(2, 4)$: Successive cancellation decoder

known as Plotkin tree [54]. Figure 2.3 details the Plotkin tree for Polar$(2, 4)$. Plotkin tree takes the input message block $\boldsymbol{m} \in \{0,1\}^n$ at the leaves and applies the "Plotkin" function at each of its internal nodes recursively to obtain the codeword $\boldsymbol{x} \in \{0,1\}^n$ at the root. The function Plotkin : $\{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^{2\ell}$, $\ell \in \mathbb{N}$, is defined as

$$\text{Plotkin}(\boldsymbol{u}, \boldsymbol{v}) \triangleq (\boldsymbol{u} \oplus \boldsymbol{v}, \boldsymbol{v}).$$

For example, in Figure 2.3, starting with the message block $\boldsymbol{m} = (0, m_2, 0, m_4)$ at the leaves, we first obtain $\boldsymbol{u} = \text{Plotkin}(0, m_2) = (m_2, m_2)$ and $\boldsymbol{v} = \text{Plotkin}(0, m_4) = (m_4, m_4)$. Applying the function once more, we obtain the codeword $\boldsymbol{x} = \text{Plotkin}(\boldsymbol{u}, \boldsymbol{v}) = (m_2 \oplus m_4, m_2 \oplus m_4, m_4, m_4)$.

2.3.2 Successive cancellation decoding

The successive-cancellation (SC) algorithm is one of the most efficient decoders for polar codes, with a decoding complexity of $O(n \log n)$. The basic principle behind the SC algorithm is to sequentially decode one message bit m_i at a time according to the conditional log-likelihood ratio (LLR), $L_i \triangleq \log(\mathbb{P}[m_i = 0|\boldsymbol{y}, \hat{\boldsymbol{m}}_{<i}]/\mathbb{P}[m_i = 1|\boldsymbol{y}, \hat{\boldsymbol{m}}_{<i}])$, given the corrupted codeword $\boldsymbol{y}$ and previous decoded bits $\hat{\boldsymbol{m}}_{<i}$ for $i \in I_k$. Figure 2.4 illustrates

this for the Polar$(2,4)$ code: for both the message bits m_2 and m_4, we compute these conditional LLRs and decode them via $\hat{m}_2 = \mathbb{1}\{L_2 < 0\}$ and $\hat{m}_4 = \mathbb{1}\{L_4 < 0\}$. Given the Plotkin tree structure, these LLRs can be efficiently computed sequentially using a depth-first-search based algorithm.

As a motivating example, let's consider the Polar$(2,2)$ code. Let the two information bits be denoted by u and v, where $u, v \in \{0,1\}$. The codeword $\boldsymbol{x} \in \{0,1\}^2$ is given by $\boldsymbol{x} = (x_1, x_2) = (u \oplus v, v)$. Let $\boldsymbol{y} \in \mathbb{R}^2$ be the corresponding noisy codeword received by the decoder. First we convert the received $\boldsymbol{y}$ into a vector of log-likelihood-ratios (LLRs), $\boldsymbol{L_y} \in \mathbb{R}^2$, which contains the soft-information about coded bits x_1 and x_2, i.e.

$$\boldsymbol{L_y} = (\boldsymbol{L_y^{(1)}}, \boldsymbol{L_y^{(2)}}) \triangleq \left(\log \frac{\mathbb{P}[y_1|x_1=0]}{\mathbb{P}[y_1|x_1=1]}, \log \frac{\mathbb{P}[y_2|x_2=0]}{\mathbb{P}[y_2|x_2=1]} \right) \in \mathbb{R}^2.$$

Once we have the soft-information about the codeword $\boldsymbol{x}$, the goal is to now obtain the same for the message bits u and v. To compute the LLRs for these information bits, SC uses the following principle: first, compute the soft-information for the left bit u to estimate $\hat{u}$. Use the decoded $\hat{u}$ to compute the soft-information for the right bit v and decode it. More concretely, we compute the LLR for the bit u as:

$$L_u = \mathrm{LSE}(\boldsymbol{L_y^{(1)}}, \boldsymbol{L_y^{(2)}}) = \log \frac{1 + e^{\boldsymbol{L_y^{(1)}} + \boldsymbol{L_y^{(2)}}}}{e^{\boldsymbol{L_y^{(1)}}} + e^{\boldsymbol{L_y^{(2)}}}} \in \mathbb{R}, \tag{2.5}$$

where $\mathrm{LSE}(a,b) \triangleq \log(1 + e^{a+b})/(e^a + e^b)$ for $a, b \in \mathbb{R}$. The expression in (2.5) follows from the fact that $u = (u \oplus v) \oplus v = x_1 \oplus x_2$ and hence the soft-information L_u can be accordingly derived from that of x_1 and x_2, i.e. $\boldsymbol{L_y}$. Now we estimate the bit as $\hat{u} = \mathbb{1}\{L_u < 0\}$. Assuming that we know the bit $u = \hat{u}$, we observe that the codeword $\boldsymbol{x} = (\hat{u} \oplus v, v)$ can be viewed as a two-repetition of v. Hence its LLR L_v is given by

$$L_v = \boldsymbol{L_y^{(1)}} \cdot (-1)^{\hat{u}} + \boldsymbol{L_y^{(2)}} \in \mathbb{R}. \tag{2.6}$$

Finally we decode the bit as $\hat{v} = \mathbb{1}\{L_v < 0\}$. To summarize, given the LLR vector $\boldsymbol{L_y}$ we first compute the LLR for the bit u, L_u, using (2.5) and decode it. Utilizing the decoded version $\hat{u}$, we compute the LLR L_v according to (2.6) and decode it.

For a more generic Polar(k,n), the underlying principle is the same: to

decode a polar codeword $\boldsymbol{x} = (\boldsymbol{u} \oplus \boldsymbol{v}, \boldsymbol{v})$, first decode the left child $\boldsymbol{u}$ and utilize this to decode the right child $\boldsymbol{v}$. This principle is recursively applied at each node of the Plotkin tree until we reach the leaves of the tree where the decoding is trivial. Given this principle, the SC algorithm for Polar$(2, 4)$, illustrated in Figure 2.4, can be mathematically expressed as (in the sequence of steps):

$$
\begin{aligned}
\boldsymbol{y} \in \mathbb{R}^4 \longrightarrow \boldsymbol{L}_{\boldsymbol{y}} &= (\boldsymbol{L}_{\boldsymbol{y}}^{(1)}, \boldsymbol{L}_{\boldsymbol{y}}^{(2)}, \boldsymbol{L}_{\boldsymbol{y}}^{(3)}, \boldsymbol{L}_{\boldsymbol{y}}^{(4)}) \in \mathbb{R}^4, \\
\boldsymbol{L}_{\boldsymbol{u}} &= (\mathrm{LSE}(\boldsymbol{L}_{\boldsymbol{y}}^{(1)}, \boldsymbol{L}_{\boldsymbol{y}}^{(3)}), \mathrm{LSE}(\boldsymbol{L}_{\boldsymbol{y}}^{(2)}, \boldsymbol{L}_{\boldsymbol{y}}^{(4)})) \in \mathbb{R}^2, \\
\text{frozen bit} \longrightarrow \hat{m}_1 &= 0, \\
L_2 &= \mathrm{LSE}(\boldsymbol{L}_{\boldsymbol{y}}^{(1)}, \boldsymbol{L}_{\boldsymbol{y}}^{(3)}) + \mathrm{LSE}(\boldsymbol{L}_{\boldsymbol{y}}^{(2)}, \boldsymbol{L}_{\boldsymbol{y}}^{(4)}) \in \mathbb{R}, \\
\hat{m}_2 &= \mathbb{1}\{L_2 < 0\} \in \{0, 1\}, \\
\hat{\boldsymbol{u}} &= (\hat{m}_2, \hat{m}_2) \in \{0, 1\}^2, \\
\boldsymbol{L}_{\boldsymbol{v}} &= (\boldsymbol{L}_{\boldsymbol{y}}^{(1)}, \boldsymbol{L}_{\boldsymbol{y}}^{(2)}) \cdot (-1)^{\hat{\boldsymbol{u}}} + (\boldsymbol{L}_{\boldsymbol{y}}^{(3)}, \boldsymbol{L}_{\boldsymbol{y}}^{(4)}) \in \mathbb{R}^2, \\
\text{frozen bit} \longrightarrow \hat{m}_3 &= 0, \\
L_4 &= \boldsymbol{L}_{\boldsymbol{v}}^{(1)} + \boldsymbol{L}_{\boldsymbol{v}}^{(2)} \in \mathbb{R}, \\
\hat{m}_4 &= \mathbb{1}\{L_4 < 0\} \in \{0, 1\}.
\end{aligned}
$$

In Figure 2.4, the above equations are succinctly represented by two sets of arrows: the black solid arrows represent the flow of soft-information from the parent node to the children and the green dotted arrows represent the flow of the decoded bit information from the children to the parent. We note that we use a simpler min-sum approximation for the function LSE that is often used in practice, i.e.

$$
\mathrm{LSE}(a, b) \approx \min(|a|, |b|)\mathrm{sign}(a)\mathrm{sign}(b), \quad a, b \in \mathbb{R}.
$$

2.3.3 Successive cancellation List decoding

SC decoding achieves the theoretically optimal performance only asymptotically, and its reliability is sub-optimal at finite blocklengths. SC decoding employs a greedy locally optimal strategy; however, these decisions may not be globally optimal. In practice, the employment of SC-list (SCL) decoding [55, 56, 57, 58] enables polar codes to achieve reliability close to capacity.

SCL improves upon its error-correction performance by maintaining a list of L candidate paths at any time step and choosing the best among them.

In fact, for a reasonably large list size L, SCL achieves MAP performance at the cost of increased complexity ($O(Ln \log n)$).

2.4 PAC codes

As described in Section 2.3, the successive cancellation decoding of polar codes is sub-optimal compared to the MAP decoding [59]. Polar codes also have poor minimum distance properties, which limits the performance of the code. One way to alleviate this issue is to add a high-rate outer code. In practice, cyclic redundancy checks (CRC) [60, 61, 62] are used in conjunction with polar codes and SCL to achieve the desired reliability.

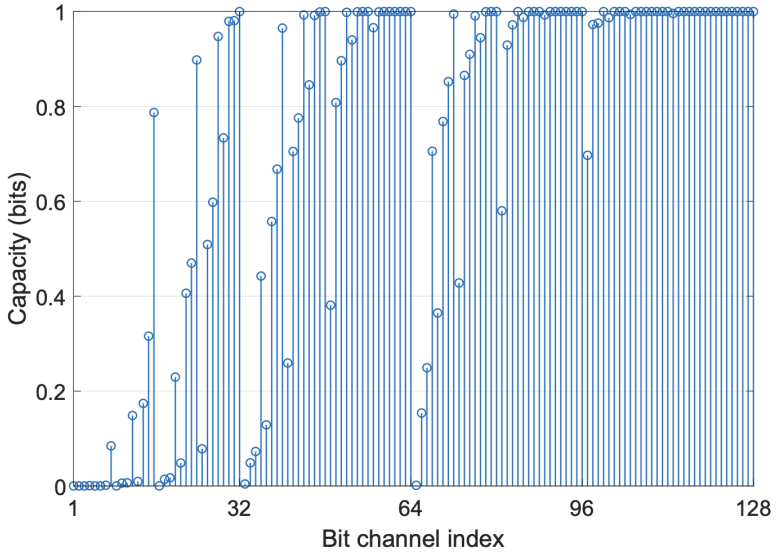


Figure 2.5: Bit channel capacities of Polar-128 code at SNR=3dB [5]

The key property of polar codes is the phenomenon of channel polarization. The bit-channels of polar codes under successive cancellation decoding are polarized. In other words, the bit channels either have a capacity of 0 or 1. But it turns out that this useful property does not hold at finite block lengths due to the existence of partially polarized bit channels. For example, the bit channel capacities for a Polar code of $N = 128$ is shown in Figure 2.5 [5]. Consequently, the capacities of the frozen bit channels are wasted.

Motivated by these drawbacks of polar codes, Arikan [5] introduces a new class of codes called Polarization-Adjusted-Convolutional (PAC) codes that match the fundamental lower bound on the performance of any code under the MAP decoding at finite-lengths [6, 63]. PAC codes shift the burden of error correction entirely to an outer code and exploit the capacities of the partially polarized channels of a polar code.

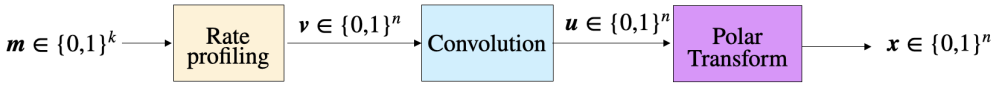


Figure 2.6: PAC encoding scheme

2.4.1 PAC encoding

PAC codes are constructed by adding a *convolutional outer code*, with an appropriate indexing I_k, before polar encoding. More formally, a rate profiling block embeds the message block $\boldsymbol{m} \in \{0,1\}^k$ is embedded into the source vector $\boldsymbol{v} \in \{0,1\}^n$. Positions $i \notin I_k$ are set to 0. The index set I_k is chosen according to the Reed-Muller (RM) rule: : compute the Hamming weights of integers $0, 1, \ldots, n-1$ and choose the top k. For instance, consider a PAC$(4, 8)$ code; $I_k = \{3, 5, 6, 7\}$. The message block $\boldsymbol{m} = \{d_0, d_1, d_2, d_3\}$ is passed through the rate profiling to obtain $\boldsymbol{v} = \{0, 0, 0, d_0, 0, d_1, d_2, d_3\}$. We obtain the input to the polar code by encoding the the rate-profiled message $\boldsymbol{v}$ via a rate-1 convolutional code, i.e.

$$\boldsymbol{u} = \boldsymbol{c} * \boldsymbol{v} \Leftrightarrow u_i = \sum_j c_j v_{i-j} \tag{2.7}$$

for some 1D convolutional kernel $\boldsymbol{c} \in \{0,1\}^\ell$. Finally we obtain the PAC codeword $\boldsymbol{x}$ by polar encoding $\boldsymbol{v}$:

$$\boldsymbol{x} = \text{PlotkinTree}(\boldsymbol{v}) \tag{2.8}$$

The PAC encoding scheme is shown in Figure 2.6.

The addition of the outer code serves to improve the distance properties of polar codes [64], which can explain the gains of finite length PAC codes over

the corresponding Polar codes under MAP decoding. Notably, PAC codes exploit all the polar bit channel capacities, enabling them to approach the BIAWGN-dispersion bound: the minimum block error rate achievable using a binary code of a given block length and rate on an AWGN channel without feedback [6].

2.4.2 Decoding of PAC codes

The PAC encoding scheme can be viewed as a binary search tree. This is an irregular tree in contrast to ones generated by convolutional codes because of the rate profiling step. This is schematically shown in Figure 2.7 for a PAC(4,8) code with $I_k = \{3, 5, 6, 7\}$. There are two branches at positions belonging to the index set I_k. In contrast, there is no branching for positions not belonging to I_k since there is only one possible value that v_i can take. The search tree branches represent all possible inputs $u_1, u_2, \cdots, u_n$ to the polar encoder. Notably, each leaf of the tree corresponds to a unique codeword. We can achieve the optimal performance by selecting the path which maximizes the likelihood, i.e., ML decoding. But since the number of leaves is exponential in K, ML decoding is intractable.

The successive cancellation algorithm described in Section 2.3.2 can be used to decode PAC codes, but it is highly sub-optimal. Heuristic tree-search methods on the PAC code tree (Figure 2.7) have achieved impressive performance. One such method is the classical Fano algorithm [7], a sequential decoding algorithm that uses backtracking [66]. It can be seen as a tree-search algorithm on the PAC search tree, which uses metrics computed using the SC algorithm. Put simply, the Fano decoder traverses down the tree if a path metric is larger than a threshold; if not, it can either backtrack and find an alternate path or reduce the threshold. Coupled with the Fano decoder, PAC codes achieve impressive results outperforming polar codes (with SCL decoder) and matching the finite-length capacity bound [6]. However, the Fano decoder has significant drawbacks like variable running time, large time complexity at low-SNRs [67], and sensitivity to the choice of hyper-parameters [68]. To overcome these issues, several non-learning techniques, such as stack/list decoding, adaptive path metrics, etc., have been proposed in the literature [65, 67, 69, 70, 71].

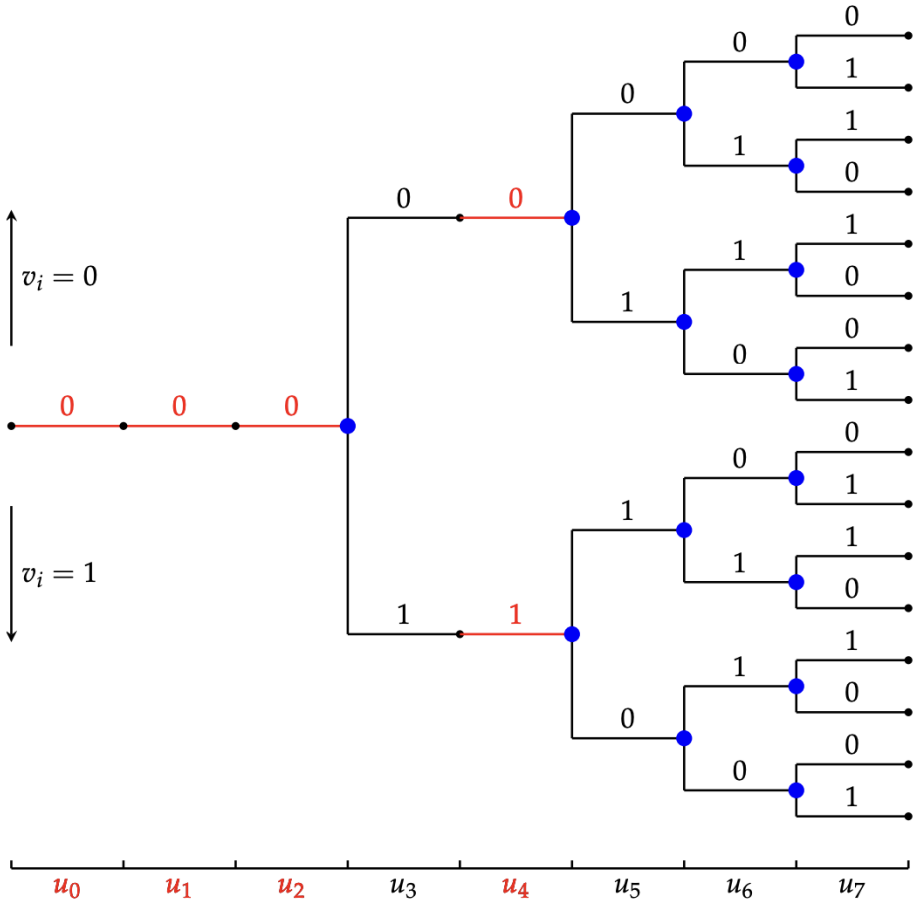


Figure 2.7: An example of the PAC search tree (reproduced from [65])

CHAPTER 3

TINYTURBO

Turbo codes (Section 2.2) have been widely used in modern communication systems and are part of 3G and 4G standards. While Turbo codes operate at near-optimal performance on the canonical additive white Gaussian noise (AWGN) channel, it is well known that the classical Turbo decoder [1] lacks robustness and performs poorly on non-AWGN channels. Thus, designing turbo decoders with high reliability and robustness is of great interest.

In the recent past, deep learning-based decoders have been shown to achieve impressive performance on Turbo decoding. Kim et al. used RNNs to train neuralBCJR [27], which is trained to imitate the BCJR algorithm. This method suffers from an error floor at high SNRs. Their follow-up work DeepTurbo [32] trained RNN and CNN-based decoders end-to-end without BCJR knowledge. These methods matched the Turbo decoding performance, along with significantly improved robustness and adaptivity to non-AWGN noise. However, these decoders have a large computational complexity, which impedes practical deployment.

Given this, more scalable approaches such as model-based DL have gained traction recently. In the context of Turbo codes, in a recent work [35], He et al. introduce *TurboNet+*, which augments learnable parameters to decoding every bit, and show that it improves the reliability of the classical Turbo decoders.

Despite the success of TurboNet+, there are three important open questions: (a) do we need all those learnable parameters, the number of which scales linearly in blocklength? (b) can we learn weights that generalize across rates, blocklengths, and codes and that are robust across channel variations? (c) what is the role of the learned weights? In this chapter, we focus on addressing these questions. Our contributions are as follows.

- We propose TinyTurbo[1], a neural augmented Turbo decoder that has 18 trainable weights. We show that TinyTurbo recovers the reliability of the state-of-the-art neural augmented Turbo decoder, namely, TurboNet+ [35] which has 720 weights, for AWGN channels.
- We show the *robustness* of TinyTurbo: TinyTurbo outperforms TurboNet+ and the classical Turbo decoder for several practical channels.
- We demonstrate the strong *generalization* of TinyTurbo: TinyTurbo trained for rate-1/3 LTE Turbo codes of blocklength 40 performs well for Turbo codes with different blocklengths, rates, and trellises (e.g., blocklength 200, rate-1/2, Turbo-757).
- Our over-the-air experiment demonstrates that TinyTurbo achieves improved reliability and efficiency compared to the classical Turbo decoders in indoor scenarios.

3.1 Weighted Max-log-MAP turbo decoder

As discussed in Section 2.2.3, approximations to the MAP algorithm such as max-log-MAP are used in practice to reduce the decoding complexity. However, this comes with a degradation in the reliability. It is well-known in the literature [72, 73, 74, 75, 76] that the max-log-MAP algorithm overestimates the LLRs of the message bits, which leads to propagation of errors in the later decoding stages. To counter this, a common technique is to scale the LLRs in Eq. (2.4). A variety of methods have been proposed to determine these scaling weights. Most existing works [72, 73, 74] propose using heuristics via off-line time-averaged estimates to obtain the best scaling weights for a specific SNR and channel setting. While Claussen et al. [75] propose to find the scaling weights that maximize the mutual information, heuristics are still needed as it is hard to estimate the mutual information. In [76], Sun and Wang use brute force search to find these parameters, which is computationally prohibitive.

[1]Code can be found at https://github.com/hebbarashwin/tinyturbo

3.2 TinyTurbo: Efficient Turbo decoding

In the previous sections, we saw the shortcomings of existing turbo decoders. Given this, it would be advantageous to have a set of scaling weights that are both robust across varying block lengths, codes, and channels. *How do we find such weights?*

In a recent work, He et al. [35] introduce TurboNet+, which can be viewed as a weight-augmented version of the max-log-MAP algorithm. TurboNet+ augments the standard max-log-MAP algorithm by adding learnable weights for each bit index $k \in [K]$ in Eq. (2.4): $L_e(u_k) = w_k^{(1)} L(u_k|\boldsymbol{y}) - w_k^{(2)} y_k^s - w_k^{(3)} L(u_k), \quad k \in [K]$.

We propose TinyTurbo, a data-driven turbo decoder where the weights in the TurboNet+ architecture are entangled across bit positions. One of the major differences with TurboNet+ and other existing techniques is that the TinyTurbo efficiently learns the optimal set of weights directly from the data using an *end-to-end loss function*. Learnt in such a purely data-driven manner, we show in Sec. 3.3 that these weights yield much better reliability results than the existing baselines and successfully generalize across various channels, block lengths, code rates, and code trellises.

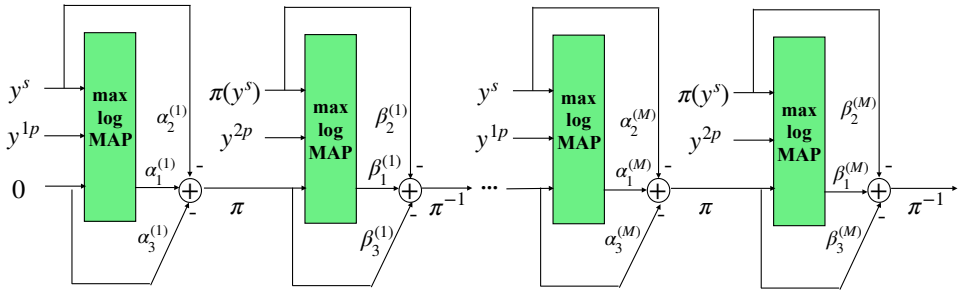


Figure 3.1: TinyTurbo decoder.

Architecture. As depicted in Figure 3.1, TinyTurbo generalizes the standard max-log-MAP algorithm by adding just *three* trainable parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3) \in \mathbb{R}^3$ in the extrinsic LLR computation, i.e.

$$L_e(u_k) = \alpha_1 L(u_k|\boldsymbol{y}) - \alpha_2 L(y_k^s) - \alpha_3 L(u_k), k \in [K]. \tag{3.1}$$

Note that the same parameters $\boldsymbol{\alpha}$ are used for all the bit indices $k \in [K]$, making it amenable to generalize across block lengths. Similarly, decoder D_2

has three additional parameters $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3) \in \mathbb{R}^3$. Thus every decoding iteration in TinyTurbo has 6 parameters $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ with the total parameters being $6M$, where M denotes the number of decoding iterations. We consider $M = 3$ in this paper. We also note that TinyTurbo can be used on top of both max-log-MAP and the MAP algorithms in Eq. 2.3. In this work, we restrict to the max-log-MAP given its computational efficiency.

Training. We propose an end-to-end loss function framework to train the TinyTurbo parameters $(\boldsymbol{\alpha}_1^M, \boldsymbol{\beta}_1^M)$. Hence these weights can be learnt directly from data alone. More precisely, let $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(B)} \in \{0,1\}^K$ denote a batch of B message blocks each of length K and let $\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \ldots, \boldsymbol{y}^{(B)} \in \mathbb{R}^N$ be the corresponding codeword-LLRs received by the TinyTurbo decoder. Let $L^M(\boldsymbol{u}^{(i)}|\boldsymbol{y}^{(i)}) \in \mathbb{R}^K$ denote the estimated LLRs of the message bits after M decoding iterations of TinyTurbo for each block $i \in [B]$, which are obtained through equation 2.4. Define the Binary Cross-Entropy (BCE) loss $L(\boldsymbol{\alpha}_1^M, \boldsymbol{\beta}_1^M)$ as

$$L(\boldsymbol{\alpha}_1^M, \boldsymbol{\beta}_1^M) \triangleq \frac{1}{B} \sum_{i=1}^{B} \sum_{k=1}^{K} u_k^{(i)} \log \sigma(L_k^M(\boldsymbol{u}^{(i)}|\boldsymbol{y}^{(i)})) + (1 - u_k^{(i)}) \log \sigma(-L_k^M(\boldsymbol{u}^{(i)}|\boldsymbol{y}^{(i)})).$$

The weights $(\boldsymbol{\alpha}_1^M, \boldsymbol{\beta}_1^M)$ are then trained by running stochastic gradient descent (SGD), or its variants like Adam [77], on the loss L. Once trained, these weights are frozen and are used directly for inference (turbo decoding). Note that the decoding complexity is comparable to that of the max-log-MAP algorithm.

As we illustrate in Section 3.3, TinyTurbo performs much better than the baselines on a variety of benchmarks.

Comparison with TurboNet+. While TinyTurbo is similar to TurboNet+, the following are the key differences: (a) TurboNet+ introduces weights for each bit index $k \in [K]$ in Eq. (2.4). As shown in Table 3.1, they have a total of $6MK$ parameters, as opposed to just $6M$ for TinyTurbo; (b) For training the weights, they consider a mean square error (MSE) between their estimated LLRs and the target BCJR-LLRs, which are obtained by running the MAP algorithm for BCJR. This introduces a computational overhead and makes the training slow as BCJR-LLRs are required for each training iteration; (c) Since the weights are learned for each bit index $k \in [K]$,

Block length	40	200
TinyTurbo	**18**	**18**
TurboNet+	720	3600

Table 3.1: Number of parameters of TinyTurbo is independent of block length

they cannot be reused for longer block lengths. In contrast, we show that by using just a $1/K$-fraction of weights and an end-to-end loss function, TinyTurbo learns a better set of weights that yield better reliability performance on various practical channels (Figures 3.6, 3.7, 3.8). . Further, our decoder generalizes better across a variety of scenarios (Figures 3.2, 3.3, 3.4, 3.5).

3.3 Results

We compare the performance of TinyTurbo with that of the standard max-log-MAP and MAP turbo decoders, and TurboNet+ [78]. We consider rate-1/3 and rate-1/2 Turbo-LTE codes of block length 40 and 200, and use the standard Quadratic Permutation Polynomial (QPP) interleaver [50].

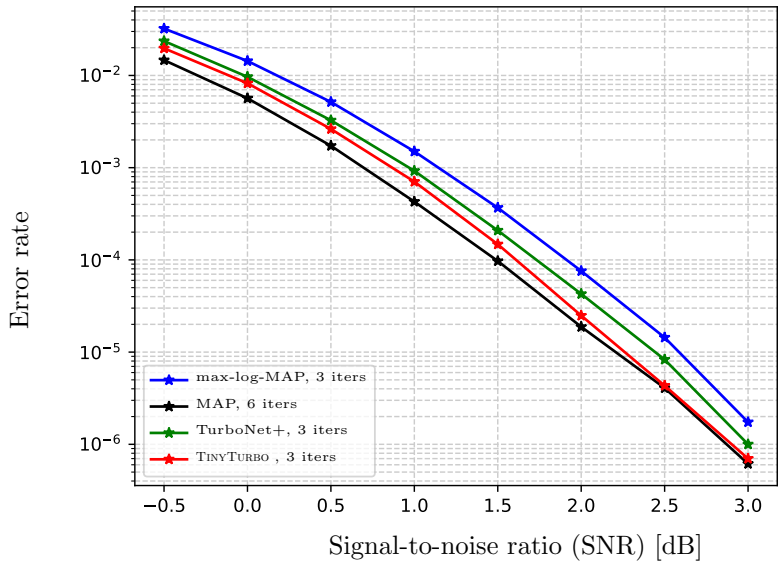


Figure 3.2: TinyTurbo trained on Turbo $(40, 132)$ outperforms the baselines and is close to MAP

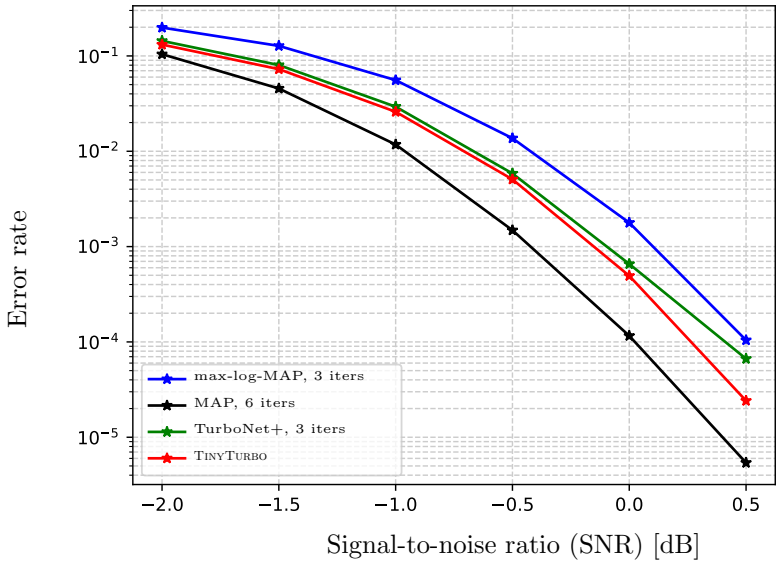


Figure 3.3: TinyTurbo trained on (40,132) generalizes to different blocklengths : Turbo(200,612)

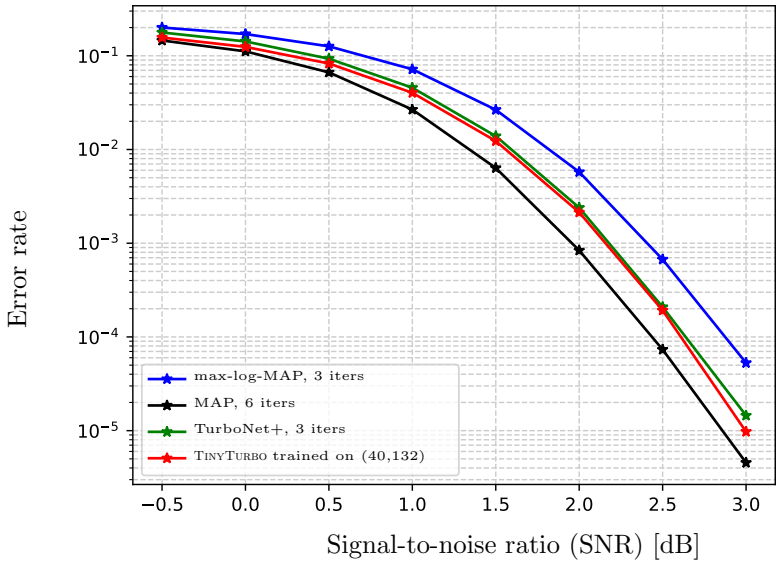


Figure 3.4: TinyTurbo generalizes to a different rate: Turbo(200, 412)

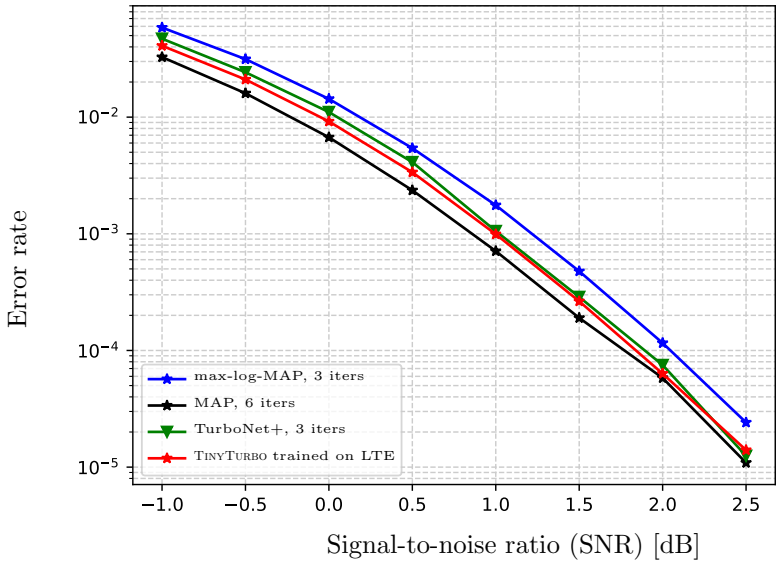


Figure 3.5: TinyTurbo generalizes to a different trellis: Turbo-757

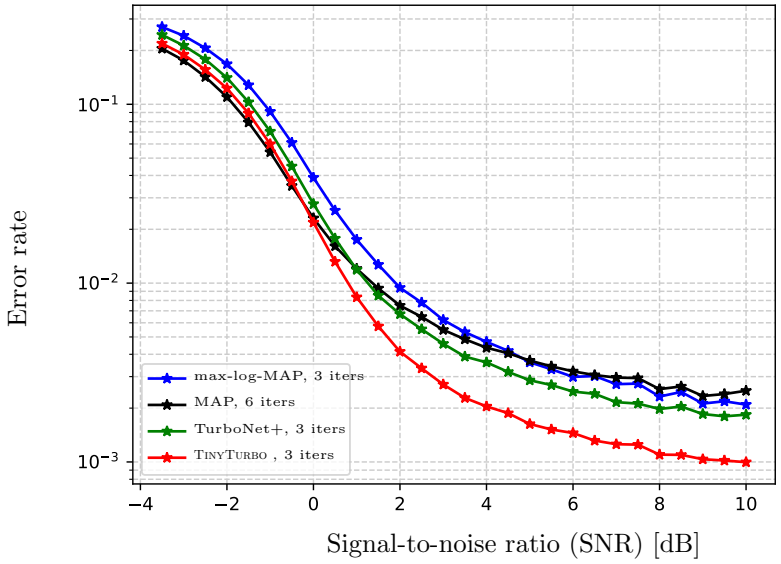


Figure 3.6: TinyTurbo exhibits strong robustness than baselines on a bursty channel

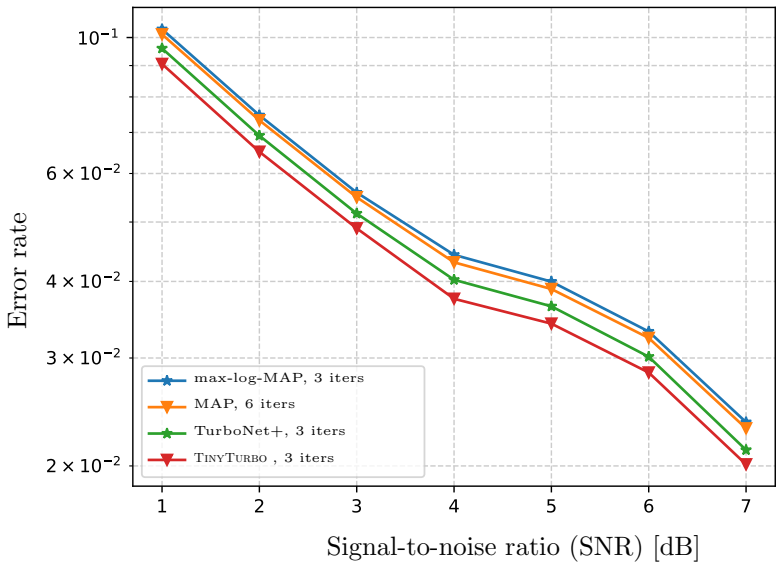


Figure 3.7: TinyTurbo exhibits robustness on EPA channel

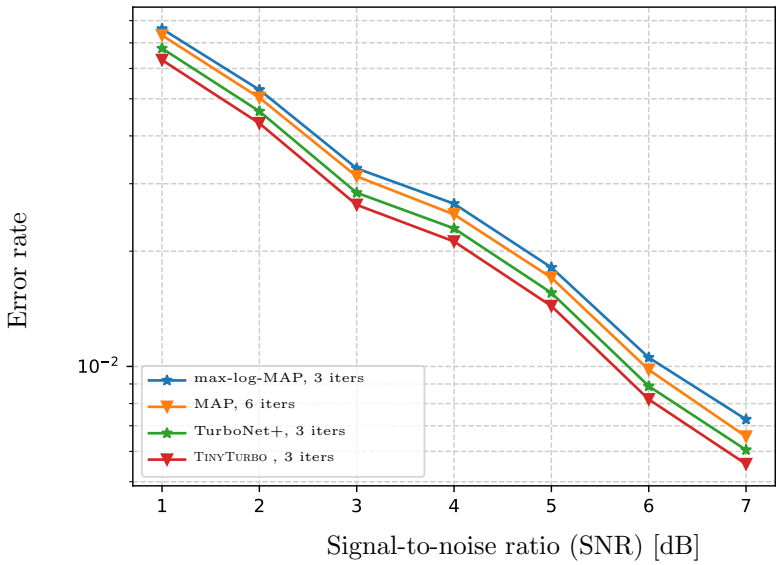


Figure 3.8: TinyTurbo exhibits robustness on EVA channel

3.3.1 AWGN channel

We train the TinyTurbo for Turbo$(40, 132)$ on the AWGN channel at -1 dB. Using these weights, we evaluate the performance of TinyTurbo on the following three codes under AWGN: Turbo$(40, 132)$, Turbo-757, and Turbo$(200, 412)$. As highlighted in Figs. 3.2, 3.4, and 3.5, TinyTurbo with 3 decoder iterations achieves a bit error rate (BER) performance close to that of the MAP decoder with 6 iterations. Further, we achieve this with a decoding complexity comparable to a max-log-MAP decoder. We also note that TinyTurbo outperforms TurboNet+ even with $40\times$ fewer parameters.

3.3.2 Generalization to other blocklengths, rates, and trellises.

One of the major shortcomings of learning-based decoders is that training them directly on larger block lengths is generally difficult due to insufficient GPU memory, and unstable training [32, 79, 42, 80]. It is thus ideal to have models that are trained on short block lengths and are reusable for longer ones. Indeed, as illustrated in Figure 3.4, we observe that TinyTurbo trained on a small code, Turbo$(40, 132)$, shows good performance when tested on a Turbo code of block length 200. Interestingly, we also notice that a TinyTurbo model trained directly on block length 200 achieves similar performance. These results highlight that our model seamlessly generalizes to longer block lengths, and we are able to bypass the bottleneck of directly training a decoder on long block lengths. Puncturing is a technique used to encode and decode codes of higher rates using standard rate 1/3 encoders and decoders. Higher rates are achieved by removing certain parity bits according to a fixed pattern. The use of puncturing improves the flexibility of a system without significantly increasing its complexity. As shown in Figure 3.4, the weights learned using a rate 1/3 Turbo code can be reused for a rate 1/2 punctured Turbo code, which demonstrates the generalizability of TinyTurbo across code rates. Surprisingly, TinyTurbo also generalizes well to different trellises; Figure 3.5 shows that TinyTurbo trained on Turbo-LTE achieves a BER performance close to MAP even on a Turbo-757 code with the generator matrix $(1, g_1(D)/g_2(D))$ with $g_1(D) = 1 + D^2$ and $g_2(D) = 1 + D + D^2$.

3.3.3 Robustness across channel variations

Here we evaluate the TinyTurbo decoder trained on AWGN on various non-AWGN settings (without any retraining). We consider both the simulated and practical channels.

We first test on a bursty channel, defined as $y = x + z + w$, where $z \sim \mathcal{N}(0, {\sigma_1}^2)$ is an AWGN noise, and $w \sim \mathcal{N}(0, {\sigma_b}^2)$ is a bursty noise with a high noise power and low probability of occurrence ρ. Here we consider $\sigma_b = 5$ and $\rho = 0.01$. Figure 3.6 highlights that TinyTurbo is significantly more robust to the bursty noise compared to the canonical Turbo decoders.

We now demonstrate the robustness of TinyTurbo on various practical channels. In particular, we consider the multi-path fading channels, as defined in the 3GPP [81]. We simulate them using the LTE toolbox in MATLAB. Specifically, we test on the Extended Pedestrian A model (EPA) and the Extended Vehicular A model (EVA) specified in the LTE standard. The EPA and EVA channels represent a low and medium delay spread environment respectively. The received signals are equalized using linear MMSE estimation before proceeding to decode. As shown in Figures 3.7 and 3.8, TinyTurbo achieves better reliability than the MAP Turbo decoder on the EPA and EVA channels.

We see that a TinyTurbo trained on a short block length is able to generalize well to longer block lengths, different rates, and different trellises. This is advantageous for edge device implementations where memory is very limited, since we do not need to store multiple models for different cases.

3.3.4 Over-the-air experiments

In this section, we use a rapid prototype system with a ZynQ SoC and an AD9361 transceiver as a testbed to evaluate the performance of our TinyTurbo algorithm in a practical setting. We implement these algorithms on the ZynQ CPU and the FPGA with an OFDM-based end-to-end communication system. Specifically, we analyze the robustness of our algorithm due to the multipath channel variations and showcase the practical viability of the algorithms on transceiver chips. In a real-time processing system, we use two of these systems as transmitter and receiver, to transmit and capture frames. The Turbo-encoded codewords were modulated and sent over

the channel after adding a preamble for synchronization and frequency offset correction. The received data was equalized and demodulated to get LLR values for the decoder. Simulations were done over different gain settings at the transmitter and the receiver to get the results for different SNRs. As demonstrated in Figure 3.9, TinyTurbo is robust to multipath fading in an indoor setting.

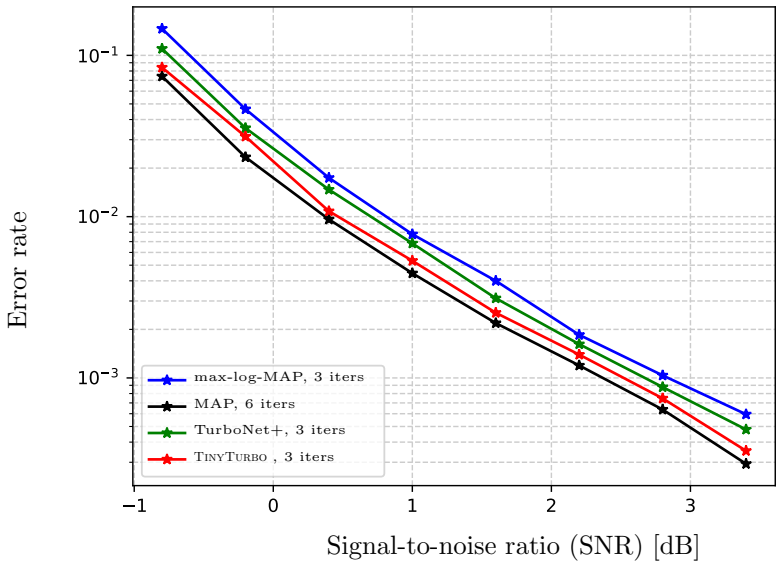


Figure 3.9: Turbo$(40, 132)$. TinyTurbo exhibits robustness in OTA experiments

3.3.5 Ablation study

The key differences between TinyTurbo and TurboNet+ are: (i) weights are shared across bit positions for TinyTurbo unlike TurboNet+, and (ii) TinyTurbo minimizes the BCE loss between the predicted LLRs and the ground truth message bits; TurboNet+ minimizes the MSE loss between the estimated LLRs and the BCJR-LLRs from the MAP algorithm. We evaluate the contributions of these components to our gains using the following ablation experiments: (i) we do not entangle weights across bit positions in TinyTurbo but train them using BCE loss. The resulting red curve in Figure 3.10 highlights that this approach has the same performance as that of TinyTurbo. (ii) we entangle weights but train them using the same procedure

as TurboNet+. The corresponding purple curve in Figure 3.10 shows similar performance as TurboNet+. Together, these experiments suggest that training the weights via the end-to-end BCE loss function approach is the key contributing factor to the gains of TinyTurbo over TurboNet+. While sharing weights did not yield any change in the performance, it nonetheless allows for a more computationally efficient decoder with a low-memory requirement.

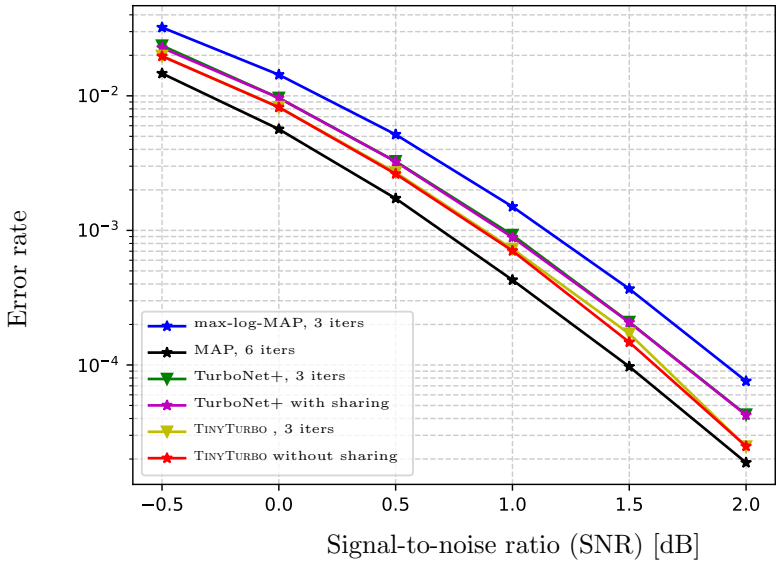


Figure 3.10: Turbo$(40, 132)$. This ablation study highlights that our gains are mainly due to the end-to-end BCE loss function.

3.4 Interpretation

	α_1	α_2	α_3	β_1	β_2	β_3
Iteration 1	0.445	0.584	1	0.641	0.779	0.662
Iteration 2	0.834	0.795	0.725	0.863	0.716	0.645
Iteration 3	0.911	0.715	0.638	0.263	0.616	0.938

Table 3.2: TinyTurbo weights

3.4.1 Interpreting TinyTurbo

In Sections 3.3 and 3.3.4, we have demonstrated that TinyTurbo [2] can achieve much better reliability than the baseline turbo decoders across a variety of channels. This raises a natural question: *where do these gains come from?* To this end, we fix the input message bits to be the all-zero vector and examine the corresponding LLR values predicted by TinyTurbo and our baselines for Turbo(40,132). In Figure 3.11, we consider the AWGN channel and plot the mean LLR together with the error bars corresponding to two standard deviations. While all the decoding algorithms have a negative mean LLR since the message bits are all-zero, only TinyTurbo and MAP decoders can keep the deviations close to/lesser than zero resulting in fewer errors. On the other hand, max-log-MAP has a significant portion of zero crossings and hence more decoding errors. These observations are consistent with our AWGN results in Figure 3.2.

We consider a simplified bursty channel model $y = x + z + w$, where x, y, z are as described in Section 3.3.3, and $w = 10.0$ only at the 57^{th} symbol in the 132-length zero codeword. Figure 3.11 illustrates that only TinyTurbo can keep the deviations contained below zero, whereas the baselines have lots of zero crossings resulting in poor decoding performance. This explains the superior performance of TinyTurbo as demonstrated in Figure 3.6.

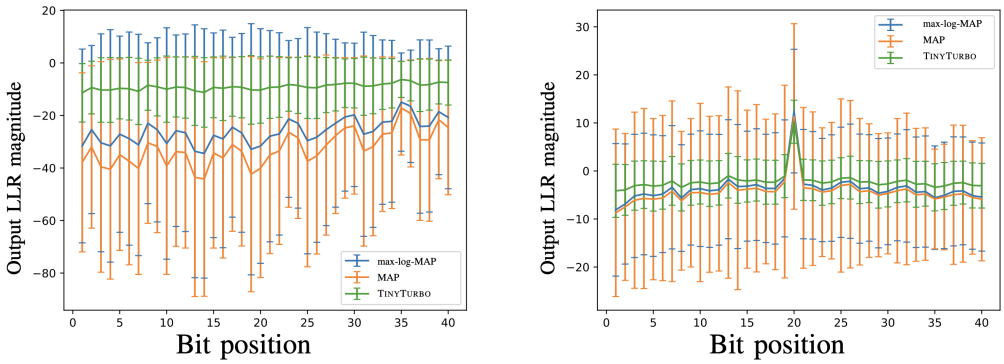


Figure 3.11: Average output LLRs on AWGN (left) and bursty (right) channels

[2] The learned weights are shown in Table 3.2.

CHAPTER 4

DQN-PAC: REINFORCEMENT LEARNING-BASED DECODING OF SHORT PAC CODES

Reinforcement learning (RL) is a paradigm of machine learning that deals with sequential decision-making. Learning proceeds without explicitly specifying the correct actions; the agent discovers optimal strategies through trial and error. Deep reinforcement learning has seen tremendous success in many sequential decision-making problems, most notably game playing [82, 83, 84].

In recent years, many works have used reinforcement learning for channel coding and decoding. Several teams [85, 86, 87] used RL to learn bit-flipping based channel decoders for linear codes. Reinforcement learning techniques were also used for sequential decoding of moderate-length LDPC codes [88, 89], polar code construction [90, 91], and belief propagation decoding of polar codes [92].

As detailed in Section 2.4.2, decoding PAC codes can be viewed as a binary search tree traversal. Existing tree-search heuristics like Fano decoding have a very high average time complexity. Reinforcement learning has been very successful in tackling tree search problems. For example, game playing can be seen as finding the best path in a decision tree. Inspired by this success, we propose DQN-PAC[1], a deep Q-network (DQN) - based algorithm to decode PAC codes. We show that this method achieves a near-MAP performance on PAC codes of block length 32 and $K \leq 14$.

4.1 Background

An agent interacts with an uncertain environment and obtains scalar rewards. The agent aims to learn to maximize the total reward received. Supervision is only provided through the reward signal. Below, we define some common terminology used in RL.

[1]Code can be found at https://github.com/hebbarashwin/dqn_pac

The agent can stay in one of many states $s \in \mathcal{S}$, and takes one of many actions $a \in \mathcal{A}$ at every time step. The agent receives a scalar reward r from the environment, and transitions to a new state s'. This is represented by a tuple (s, a, s', r) known as a *transition step*. The state transition function determines the probability of the agent moving to a state s' from s after taking an action a. The agent chooses its action at state s based on its policy π. The policy can be deterministic : $\pi(s) = a$, or stochastic : $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$. Reinforcement learning problems are modeled as Markov decision processes (MDP). MDPs consist of states that satisfy the Markov property: the future only depends on the current state and not the past history.

$$\mathbb{P}[S_{t+1}|S_t, St-1, \cdots, S_1] = \mathbb{P}[S_{t+1}|S_t]$$

.

The reward function $R_t(s, a)$ is the expected value of the reward received if action a is taken at state s.

$$R_t(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

The goal of RL is to learn an optimal policy $\pi^\star$ that maximizes the cumulative reward.

The return G_t is the future reward accumulated after the k^{th} timestep:

$$G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Here, γ is a discount factor that determines the importance of rewards in the distant future relative to immediate rewards.

The Q-function $Q(s, t)$ is a function that estimates the quality of a state action pair (s, a). It is the expected return starting from s if action a is chosen.

$$Q(s, t) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

The Q function satisfies the Bellman equations, which decomposes it into a sum of the immediate reward and the future values.

$$Q(s, a) = \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a)|S_t = s, A_t = a]$$

4.2 DQN decoding

Q-learning is an off-policy RL algorithm that seeks to find the optimal action at the current state. This is done by learning the optimal Q-values for all state-action pairs. Given this, at inference, the optimal policy can be followed by choosing an action that maximizes the Q function at the current state.

$$\pi^{\star}(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a)$$

The sequence of states, actions, and rewards until a terminal state is reached is called an *episode*. Within an episode, the Q-learning algorithm is as follows:

1. Begin at the initial state S_0 at $t = 0$.
2. At timestep t, we rake an action according to an ϵ -greedy policy. This is a policy chosen to address the exploration-exploitation tradeoff. The agent chooses a random action with probability ϵ, and chooses the greedy action $a_t = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$ with probability $1 - \epsilon$.
3. The reward R_t is observed and the agent moves to the next state S_{t+1} according to the transition function.
4. Update the Q-function:

$$Q(S_t, A_t) \leftarrow (1 - \alpha)Q(S_t, A_t) + \alpha(R_t + \max_{a \in \mathcal{A}} Q(S_{t+1}, a)$$

5. $t+ = 1$. Go to step 2.

We can employ a Q-table to memorize the optimal Q-values for all state-action pairs. But this is infeasible if the state or action spaces are large. We can circumvent this issue by training a neural network $Q_\theta(s, a)$ to approximate the Q-values. Since Q-values satisfy the Bellman equation, we train the parameters θ such that the mean squared error between the $Q_\theta(s, a)$ and the Bellman equation target is minimized:

$$\mathcal{L} = (Q_\theta(S_t, A_t) - (R_t + \max_{a \in \mathcal{A}} Q_\theta(S_{t+1}, a)))^2 \tag{4.1}$$

We use the DQN algorithm [93] that stabilizes deep Q-learning training. We describe some crucial training methods [94] that allow stable training and faster convergence.

- **Periodically updated target** The target in Q-learning is given by the Bellman equation. At a non-terminal state S_t, the target $= (R_t + \max_{a \in \mathcal{A}} Q_\theta(S_{t+1}, a)$. In this learning paradigm, the target changes at each update step, leading to unstable training. To tackle this, we use two copies of the DQN, termed the policy network Q_θ and the target network Q_θ^T, which is kept frozen. The parameters of the target network are periodically updated to the policy network. The target network is used to compute the Bellman target, while the policy network is updated during training.

- **Experience replay** The high correlation between samples gathered during exploration leads to unstable training. The DQN algorithm tackles this by disentangling the learning phase from the experience gathering phase. The agent's experiences $e = (s_t, a_t, r_t, s_{t+1})$ are stored in a replay buffer B. A minibatch is sampled from the buffer to learn the optimal policy. The use of experience replay and off-policy training improve data efficiency since a sample can be used multiple times.

4.3 DQN-PAC

In this section, we discuss the formulation of PAC decoding within the RL framework. Consider the decoding of a PAC(k, n) code. Our objective is to estimate the message bits $\hat{m} = (\hat{m}_0, \hat{m}_1, \cdots \hat{m}_{k-1})$, or equivalently the rate profiled vector $\hat{v} = (\hat{v}_0, \hat{v}_1, \cdots \hat{v}_{n-1})$ given the received vector $\boldsymbol{y}$. We pose this as a problem of finding the best path on the PAC search tree; which is a surrogate for minimizing the block error rate (BLER).

At time step t, we define our state to be the concatenation of the received vector and the previously decoded bits:

$$s_t = (\boldsymbol{y}, \hat{v}_0^{t-1}) \in \mathbb{R}^n \times \{\pm 1\}^t$$

. The actions are the estimated decoded message bits:

$$a_t = \hat{v}_t = \begin{cases} \in \{\pm 1\}, & t \in I_t. \\ +1, & t \notin I_t. \end{cases}$$

Recall that at timesteps not in the index set, the bit v_k is always set to $+1$. So the action is known to be $+1$. Otherwise, the agent has two options to choose from. The state transition can be viewed as appending the latest action to the previous state:

$$s_{t+1} = (s_t, a_t) = (y, \hat{v}_0^{t-1})$$

We define our reward to be the maximum likelihood (ML) metric:

$$\text{ML - metric} = \mathbb{P}[\hat{v}_t = a_t | y, \hat{v}_0^{t-1}]$$

We can compute this metric efficiently using the successive cancellation algorithm on the polar tree. We use the fact that u_t can be uniquely determined given the decoded bits v_0^{t-1} using (2.7). SC gives us the LLRs corresponding to u_t.

$$\begin{aligned} R_t(s_t, a_t) &= \mathbb{P}[\hat{v}_t = a_t | y, \hat{v}_0^{t-1}] \\ &= \mathbb{P}[\hat{u}_t = a_t | y, \hat{u}_0^{t-1}] \\ &= \log \sigma(LLR_t \cdot \hat{u}_t) \end{aligned}$$

The PAC decoding objective is to find the optimal set of actions that maximizes the cumulative reward:

$$\text{maximize}_{(a_0, a_1, \cdots a_{t-1})} \sum_{t=0}^{n-1} R_t(s_t, a_t)$$

The discount factor $\gamma = 1$. In this formulation, the state space is continuous, while the action space is discrete. The state transition function is deterministic, i.e., given the same state and action, the agent always transitions to the same next state. We use the DQN formulation to learn the decoding objective.

4.4 Training

We use the DQN framework explained in Section 4.2 to train the decoding agent. We parametrize the Q-network by a fully connected neural network (FCNN). We first initialize the replay buffer with E_g ground-truth experiences and E_i initial episodes before starting the training. We use an ϵ-greedy exploration strategy, with a decaying ϵ as training progresses. Each episode consists of k decoded bits (actions by the decoding agent) and terminates when all bits are decoded. At each exploration step, the experience $e = (s_t, a_t, r_t, s_{t+1})$ is pushed into the replay buffer. We use the DQN framework explained in Section 4.2 to train the decoding agent.

The number of leaves in the decoding tree increase exponentially with increasing k. Consequently, we require longer training to achieve a good performance. One of the methods that was critical to improving the training sample efficiency was prioritized experience replay [95, 94]. In this method, the experiences in the replay buffer are sampled with a probability proportional to the last encountered temporal difference error:

$$p_t \propto |R_{t+1} + \max_{a'} Q_\theta^T(S_{t+1}, a') - Q_\theta(S_t, A_t)|$$

. Thus, since harder examples are shown more often, the training proceeds significantly faster.

The space of codewords increases exponentially with increase in k. This makes the exploration much harder, and consequently the training becomes very slow. To scale to larger k, we propose a curriculum training approach. Our key observation is that a PAC(k, n) code subsumes all the codewords of lower-rate subcodes PAC$(i, n), 1 \leq i \leq k$. We use this nested property to progressively train these subcodes. For instance, the training of a PAC$(14, 32)$ DQN proceeds as follows:

$$\text{DQN-PAC}(8, 32) \rightarrow \cdots \rightarrow \text{DQN-PAC}(13, 32) \rightarrow \text{DQN-PAC}(14, 32)$$

We define the curriculum to train a PAC(k, n) code by progressively training the PAC codes of smaller k, and using those trained weights to initialize the next curriculum step. We found that such a curriculum training strategy was essential to obtaining faster convergence.

4.5 Results

In this section, we evaluate the DQN-based PAC decoder on codes of block length 32 on the AWGN channel. We use the Fano decoder (Section 2.4.2), which operates close to the optimal MAP performance, as our baseline.

Figure 4.1 highlights that the DQN decoding achieves a near-MAP performance on PAC$(12, 32)$ codes. A similar observation is seen in Figure 4.2, where the DQN decoding achieves a reliability close to that of the Fano algorithm. Notably, the curriculum training strategy enabled us to achieve a much better reliability.

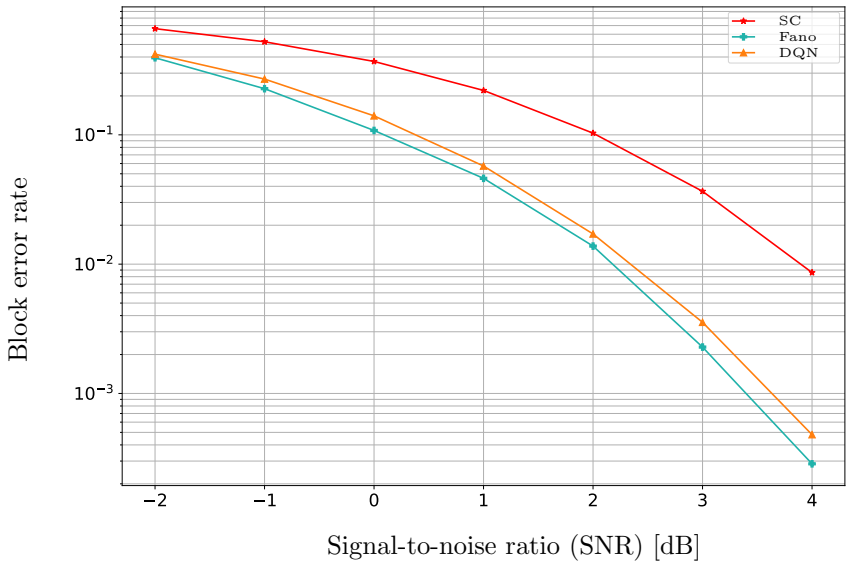


Figure 4.1: DQN achieves near-MAP performance on PAC$(12, 32)$

Supervised learning-based channel decoding methods typically optimize the bit error rate using suitable surrogates like the MSE between the message and estimated message bits (Section 2.1). However, in many practical application we are interested in optimizing the block error rate. DQN-PAC provides a method to optimize the block error rate of channel decoding. Although we obtain good results on decoding of short length polar codes, the training becomes very expensive as we move to longer codes. The curriculum training strategy of DQN-PAC is not scalable to larger codes; every curriculum step required > 50 hours of training on a single GPU.

We also note that we are providing very weak supervision using the reward

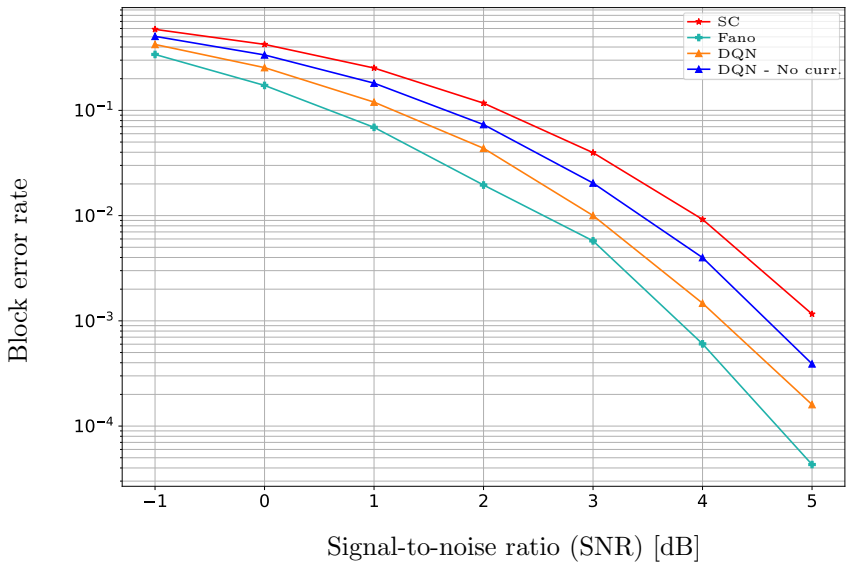


Figure 4.2: DQN achieves reasonably good reliability on PAC$(14, 32)$ decoding

signal that uses LLRs from the successive cancellation algorithm. We have access to the ground truth labels, which can be leveraged to provide stronger supervision during training. With access to a higher degree of supervision, we can hope to achieve a more sample-efficient learning algorithm.

The DQN-PAC decoder described in this chapter used an RL-based tree-search approach to learn to minimize the block error rate of PAC decoding. In Section 5.1 we describe a neural decoder trained via supervised learning to decode Polar codes and PAC codes. This approach in conjunction with curriculum learning proves to be a more scalable approach.

CHAPTER 5

CRISP: CURRICULUM-BASED SEQUENTIAL NEURAL DECODERS FOR POLAR AND PAC CODES

The polar family exhibits several crucial information-theoretic properties; practical finite-length performance, however, depends on high complexity decoders. This search for the design of efficient and reliable decoders for the Polar family is the focus of substantial research in the past decade. **(a) Polar codes:** The classical successive cancellation (SC) decoder achieves information-theoretic capacity asymptotically, but performs poorly at finite blocklengths compared to the optimal maximum a posteriori (MAP) decoder [5]. To improve upon the reliability of SC, several polar decoders have been proposed in the literature. One such notable result is the celebrated Successive-Cancellation-with-List (SCL) decoder [55]. SCL improves upon the reliability of SC and approaches that of the MAP with increasing list size (and complexity). **(b) PAC codes:** The sequential "Fano decoder" [7] allows PAC codes to perform information-theoretically near-optimally; however, the decoding time is long and variable [8]. Although SC is efficient, $O(n \log n)$, the performance with PAC codes is significantly worse than that of the Fano decoder. Several works [65, 67, 69, 70, 71, 96] propose ameliorations; it is safe to say that constructing efficient and reliable decoders for the Polar family is an active area of research and of utmost practical interest given the advent of Polar codes in 5G wireless cellular standards. The design of efficient and reliable decoders for the Polar family is the focus of this chapter.

The DQN based decoders achieved good performance for short codes, however it is computationally prohibitive to scale to larger block lengths. One drawback of the RL-based approach is that we are providing a very weak supervision in the form of rewards. However, we have access to the ground truth message bits, which can act as a much stronger supervision signal. In this section, we describe an RNN-based sequential decoder CRISP[1], for Polar and PAC codes which is trained via supervised learning. We describe our meth-

[1]Code can be found at https://github.com/hebbarashwin/crisp

ods in the context of polar decoding; the same ideas readily extend to PAC decoding. We show that CRISP outperforms existing baselines and attains near-MAP reliabilty on Polar$(22, 64)$, Polar$(16, 32)$ and PAC$(16, 32)$ codes whilst having an efficient decoder. Our key observation is that a Polar(k, n) code subsumes all the codewords of lower-rate subcodes Polar$(i, n), 1 \leq i \leq k$ (Section 2.3). Capitalizing on this nested property, we design a principled curriculum of training on these subcodes which enables CRISP to attain near-optimal performance (Section 5.2).

We design CRISP, a curriculum-learning-based sequential neural decoder for polar codes that strictly outperforms the SC algorithm and existing baselines. CRISP uses a sequential RNN decoder, powered by gated recurrent units (GRU) [97], to decode one bit at a time. Instead of standard training techniques, we design a novel curriculum to train the RNN to learn good decoders. Figure 5.2 illustrates our approach.

Earlier works on designing neural polar decoders [98] used off-the-shelf neural architectures. These were only able to decode codes of small block-length (≤ 16) [48, 99, 100, 101]. Later works augmented belief propagation decoding [102, 103, 104, 91], and SC-flip decoding [105, 106] with neural components and improved performance. In [107, 108], the authors replace sub-components of the existing SC decoder with NNs to scale decoding to longer lengths. However, these methods fail to give reasonable reliability gains compared to SC. In contrast, we use curriculum learning to train neural decoders and show non-trivial gains over SC performance. Our approach is closest to that of Lee et al. [109], who consider a progressive training of polar codes using the naive curriculum.

5.1 CRISP decoder

We use the Polar$(2, 4)$ code as a guiding example to illustrate our decoder (Figure 5.1). This code has two message bits (m_2, m_4) and the message block is $\boldsymbol{m} = (0, m_2, 0, m_4)$. Upon encoding it to the polar codeword $\boldsymbol{x} \in \{\pm 1\}^4$ and receiving its noisy version $\boldsymbol{y} \in \mathbb{R}^4$, the decoder estimates the message as $\hat{\boldsymbol{m}} = (0, \hat{m}_2, 0, \hat{m}_4)$. Similar to SC, CRISP uses the sequential paradigm of decoding one bit $\hat{m}_i$ at a time by capitalizing on the previous decoded bits $\hat{\boldsymbol{m}}_{<i}$ and $\boldsymbol{y}$. To that end, we parameterize the bit estimate $\hat{m}_i$ conditioned

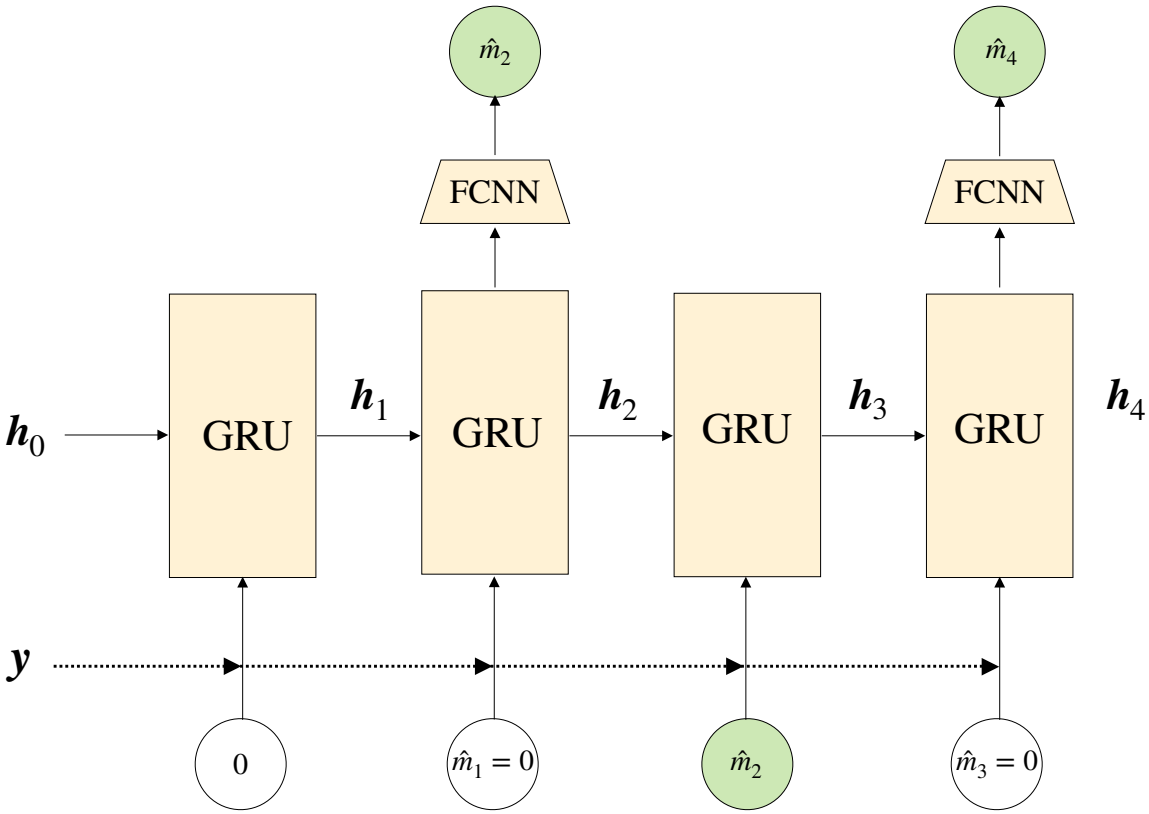


Figure 5.1: CRISP decoder for Polar$(2, 4)$.

on the past as a fully connected neural network (FCNN) that takes the hidden state $\boldsymbol{h}_i$ as its input. Here $\boldsymbol{h}_i$ denotes the hidden state of the GRU that implicitly encodes this past information $(\hat{\boldsymbol{m}}_{<i}, \boldsymbol{y})$ via GRU's recurrence equation, i.e.

$$\boldsymbol{h}_i = \text{GRU}_\theta(\boldsymbol{h}_{i-1}, \hat{m}_{i-1}, \boldsymbol{y}), \quad i \in \{1, 2, 3, 4\}, \tag{5.1}$$

$$\hat{m}_i | \boldsymbol{y}, \hat{\boldsymbol{m}}_{<i} = \text{FCNN}_\theta(\boldsymbol{h}_i), \quad i \in \{2, 4\}, \tag{5.2}$$

where θ denotes the FCNN and GRU parameters jointly. Henceforth we refer to our decoder as either CRISP or CRISP$_\theta$. Note that while the RNN is unrolled for $n = 4$ time steps ((5.1)), we only estimate bits at $k = 2$ information indices, i.e. $\hat{m}_2$ and $\hat{m}_4$ ((5.2)). A key drawback of SC is that a bit error at a position i can contribute to the future bit errors $(> i)$, and it does not have a feedback mechanism to correct these error events. On the other hand, owing to the RNN's recurrence relation ((5.1)), CRISP can learn to

correct these mistakes through the gradient it receives (via backpropagation through time) during training.

5.1.1 Training

Given the decoding architecture of CRISP in Figure 5.1, a natural approach to train its parameters via supervised learning is to use a joint MSE loss function for both the bits $(\hat{m}_2, \hat{m}_4)$: $\text{MSE}(\hat{m}_2, \hat{m}_4) = (\hat{m}_2(\theta) - m_2)^2 + (\hat{m}_4(\theta) - m_4)^2$. However, as we highlight in Section 5.2, such an approach learns to fail better decoders than SC and gets stuck at local minima. To address this issue, we propose a curriculum-learning based approach to train the RNN parameters.

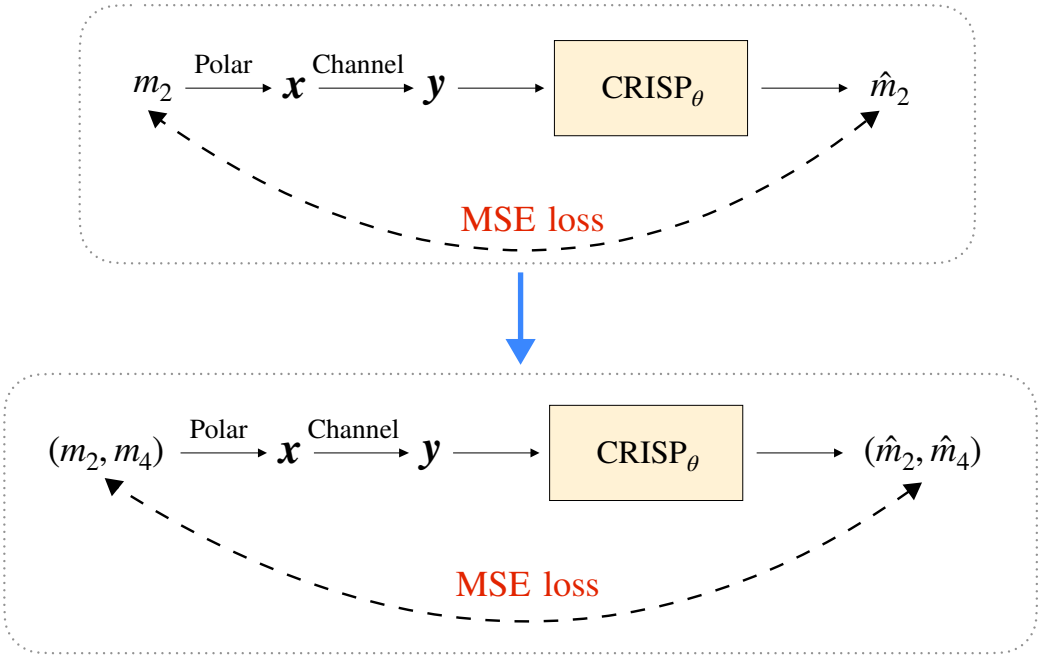


Figure 5.2: Curriculum to train CRISP

The key idea behind our curriculum training of CRISP is to decompose the problem of joint estimation of bits $(\hat{m}_2, \hat{m}_4)$ into a sequence of sub-problems with increasing difficulty: start with learning to estimate only the first bit $(\hat{m}_2)$ and progressively add one new message bit at each curriculum step $(\hat{m}_4)$ until we estimate the full message block $\boldsymbol{m} = (\hat{m}_2, \hat{m}_4)$. We freeze all the non-trainable message bits to zero during any curriculum step. In other words, in the first step, we freeze the bit m_4 and train the decoder only to

estimate the bit $\hat{m}_2$ (i.e. the subcode corresponding to $k = 1$):

$$(m_2, m_4 = 0) \to \boldsymbol{m} = (0, m_2, 0, 0) \xrightarrow{\text{Polar}} \boldsymbol{x} \xrightarrow{\text{Channel}} \boldsymbol{y} \xrightarrow{\text{CRISP}_\theta} \hat{m}_2. \tag{5.3}$$

We use this trained θ as an initialization for the next task of estimating both the bits $(\hat{m}_2, \hat{m}_4)$:

$$(m_2, m_4) \to \boldsymbol{m} = (0, m_2, 0, m_4) \xrightarrow{\text{Polar}} \boldsymbol{x} \xrightarrow{\text{Channel}} \boldsymbol{y} \xrightarrow{\text{CRISP}_\theta} (\hat{m}_2, \hat{m}_4). \tag{5.4}$$

Figure 5.2 illustrates this curriculum-learning approach. We note that the knowledge of decoding $\hat{m}_2$ when $m_4 = 0$ ((5.3)) serves as a good initialization when we learn to decode $\hat{m}_2$ for a general $m_4 \in \{0, 1\}$ ((5.4)). With such a curriculum aided training, we show that the CRISP decoder outperforms the existing baselines and attains near-optimal performance for a variety of blocklengths and codes.

Left-to-Right (L2R) curriculum for Polar(k, n). For a general Polar(k, n) code, we follow a similar curriculum to train CRISP$_\theta$. Denoting the index set by $I_k = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ in the increasing order of indices $i_1 < i_2 < \ldots < i_k$, our curriculum is given by: Train θ on $\hat{m}_{i_1} \to$ Train θ on $(\hat{m}_{i_1}, \hat{m}_{i_2}) \to \ldots \to$ Train θ on $(\hat{m}_{i_1}, \ldots, \hat{m}_{i_k})$. We term this curriculum *Left-to-Right (L2R)*. The anti-curriculum *R2L* refers to progressively training in the decreasing order of the indices in I_k.

5.2 Results

In this section, we present numerical results for the CRISP decoder on the Polar code family.

The optimal channel decoder is given by the MAP estimator:

$$\hat{\boldsymbol{m}} = \arg \max_{\boldsymbol{m} \in \{0,1\}^k} \mathbb{P}[\boldsymbol{m}|\boldsymbol{y}]$$

, whose complexity grows exponentially in k and is computationally infeasible. Given this, we compare our CRISP decoder with SCL [55], which has near-MAP performance for a large L, along with the SC baseline. Among learning-based decoders, we choose the state-of-the-art Neural-Successive-Cancellation (NSC) as our baseline [108]. NSC is a model-based DL ap-

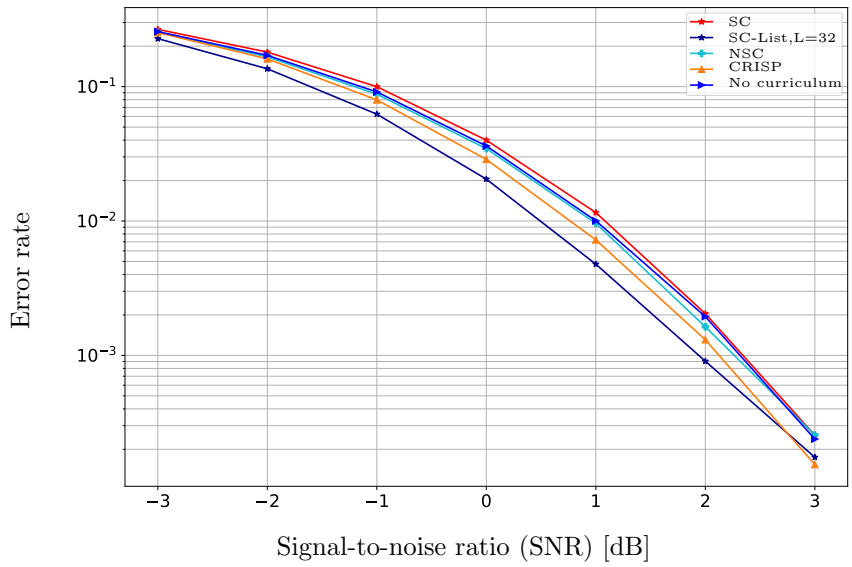


Figure 5.3: CRISP achieves near-MAP reliability for Polar$(22, 64)$ code on the AWGN channel

proach that leverages the structure of the SC tree. In this method, sub-trees of depth 4 are replaced by neural networks trained to decode polar codes of block length 16. While the original NSC uses a sub-optimal training procedure with SC probabilities as the target, we consider an improved version with end-to-end training (Figure 2.1) for a fair comparison. We also include the performance of CRISP trained directly without the curriculum. Both these baselines have the same number of parameters as CRISP.

Figure 5.3 highlights that the CRISP decoder outperforms the existing baselines and attains near-MAP performance over a wide range of SNRs for the Polar$(22, 64)$ code. NSC is restricted by the structure of the SC decoding tree, and does not outperform SC. Figure 5.4 illustrates the mechanism behind our gains at 0dB: the curriculum-guided CRISP slowly improves upon the overall BER (over the 22 bits) during the training and eventually achieves much better performance than SC and other baselines. In contrast, the decoder trained from scratch makes a big initial gain but gets stuck at local minima and only achieves a marginal improvement over SC. We observe a similar trend for Polar$(16, 32)$ code in Figure 5.5, where CRISP achieves near-MAP performance. We posit that aided by a good curriculum, CRISP avoids getting stuck at bad local minima and converges to better minima in

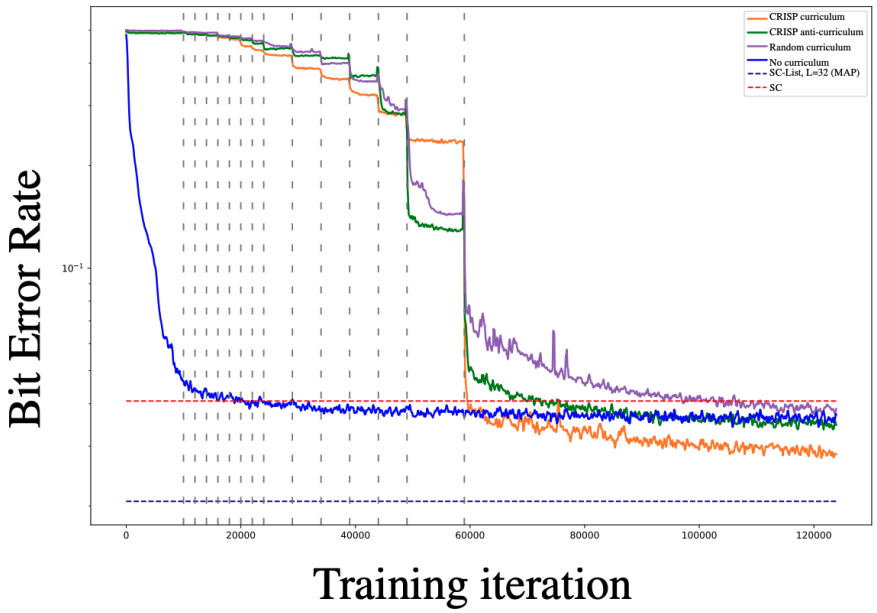


Figure 5.4: Our proposed curriculum is crucial for the gains CRISP attains over the baselines

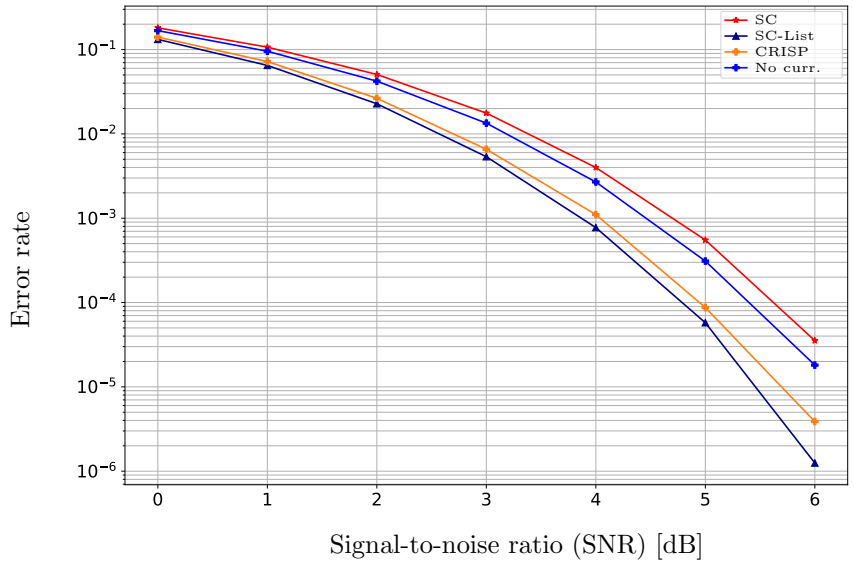
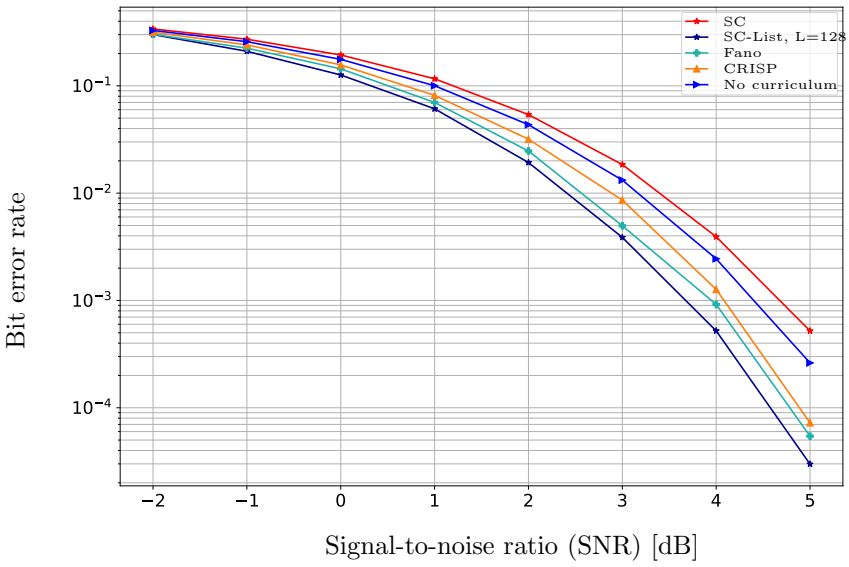


Figure 5.5: Polar$(16, 32)$



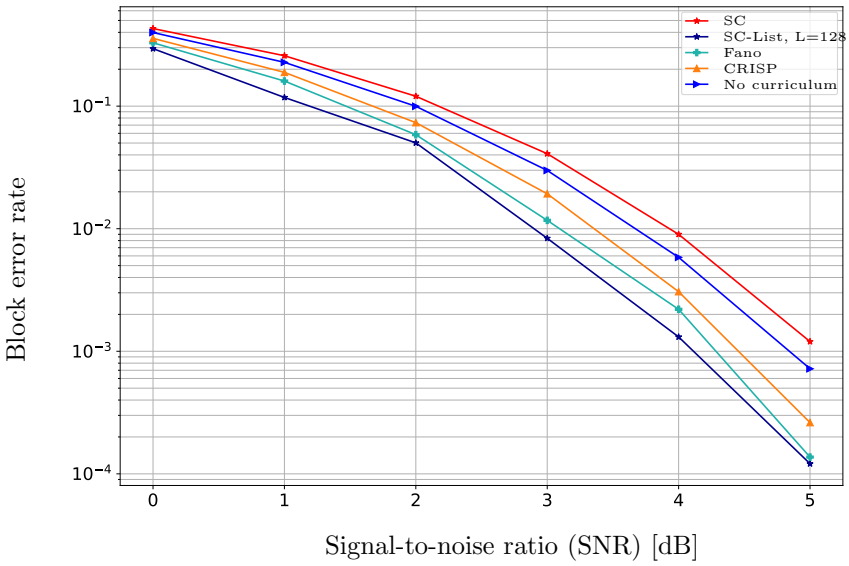


Figure 5.6: PAC$(16, 32)$

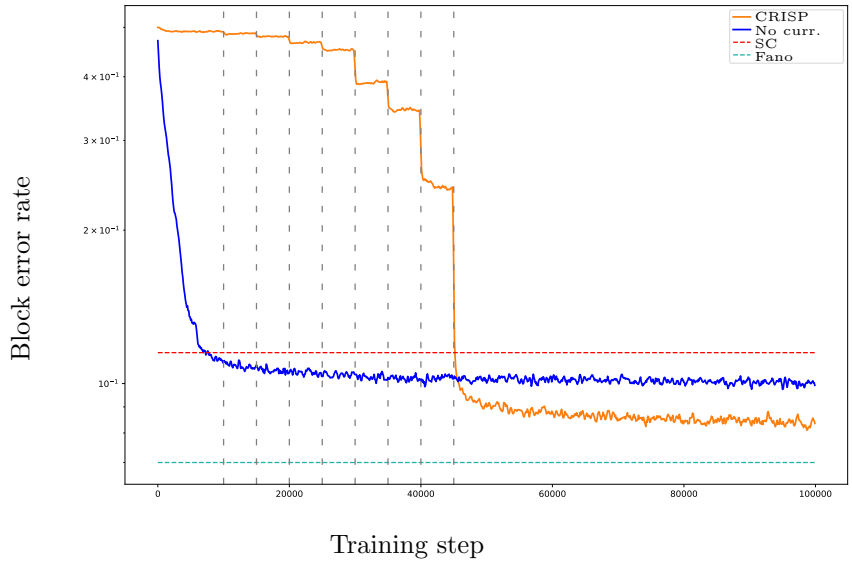


Figure 5.7: PAC(16,32) - Progressive train

the optimization landscape.

5.3 CRISP PAC decoder

We demonstrate that the CRISP decoder achieves a good decoding performance for PAC codes. In contrast to the DQN approach, the RNN-based decoder for PAC codes is trained in a supervised manner, exploiting the ground truth messages. We use the same architecture and L2R curriculum described above to decode PAC codes.

Figure 5.6 highlights that the CRISP decoder achieves near-MAP performance for the PAC$(16, 32)$ code. While Fano decoding achieves similar reliability, it is inherently non-parallelizable. In contrast, neural decoders allow for batching and are highly optimized on GPUs, and can achieve a higher throughput: at SNR $= 1$ dB, Fano [8] takes 6.3 minutes on average to decode 100K samples whereas CRISP only takes 2 seconds (on GTX 1080 Ti GPU).

5.4 Implementation details

5.4.1 Architecture

We use a 2-layer GRU with a hidden state size of 512. The output at each timestep is obtained through a fully connected layer (as shown in Figure 5.1). The network has 2.5M and 600K parameters for block lengths 64 and 32. As shown in Figure 5.8, 2-layer-LSTM and 3-layer-GRU models achieve similar performance. We choose a 2-layer GRU for our experiments since it allows for faster training and has fewer parameters.

5.4.2 Training.

We also note that it is a standard practice to use *teacher forcing* to train sequential models [110]: during training, as opposed to feeding the model prediction $\hat{m}_i$ as an input for the next time step, the ground truth message bit m_i is provided as an input to the model instead (Figure 5.1). *Student forcing* refers to using the same $\hat{m}_i$ as an input. We found that teacher forcing gives a better final performance in terms of both BER and BLER, whereas student forcing only provides gains in the BER reliability (Figure 5.9). We observed that student forced training achieved sub-optimal performance for larger block lengths.

Empirically, we observed that the number of iterations spent on training each intermediate subcode of the curriculum is not critical to the performance of the final model (Figure 5.10). To train CRISP for Polar(22,64), we use the following curriculum schedule: Train each subcode for 2000 iterations, and finally train the full code until convergence with a decaying learning rate. This training schedule required 13-15 hours of training on a GTX 1080Ti GPU.

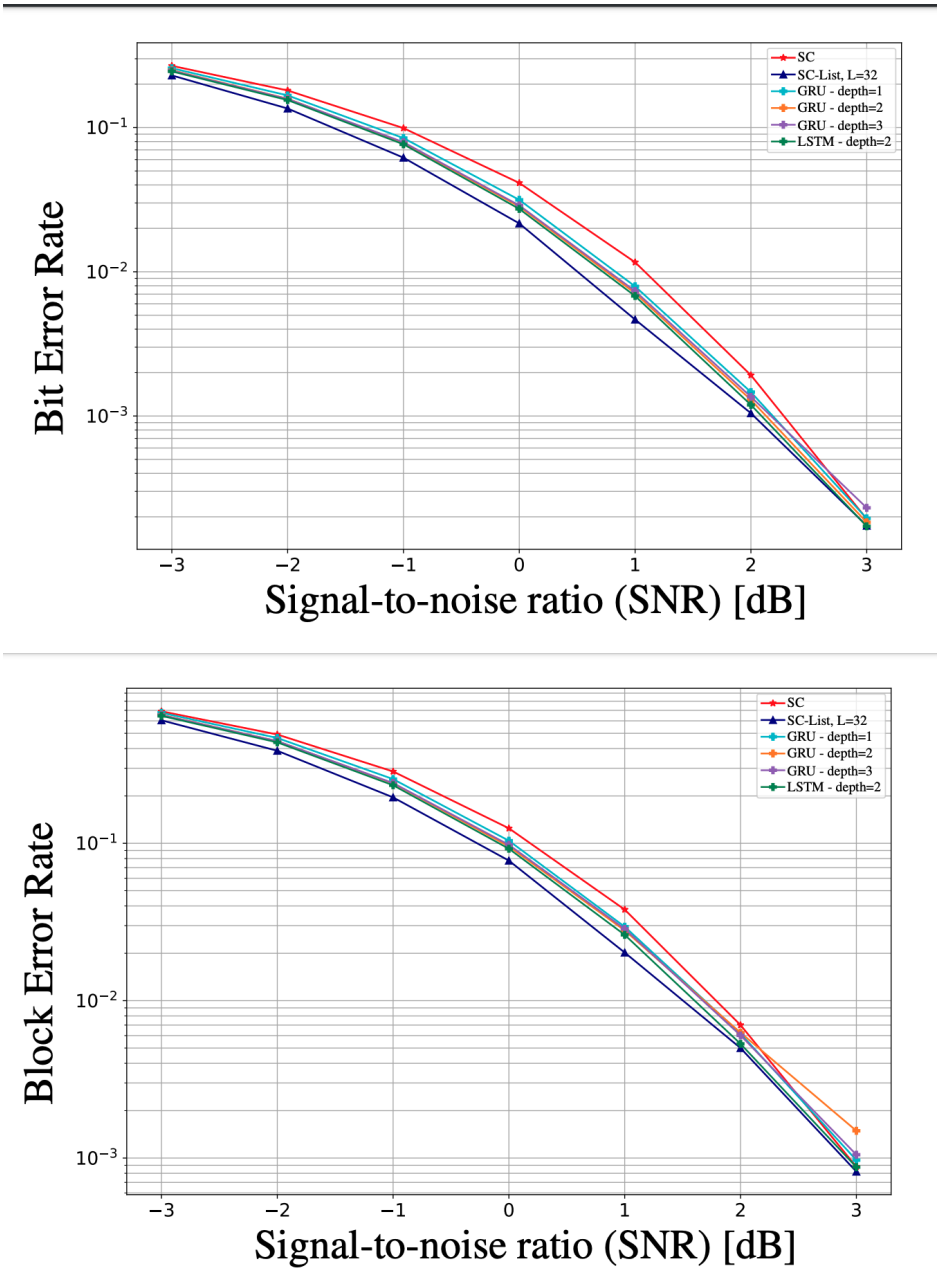


Figure 5.8: Polar$(22, 64)$: LSTMs and GRUs achieve similar reliability.

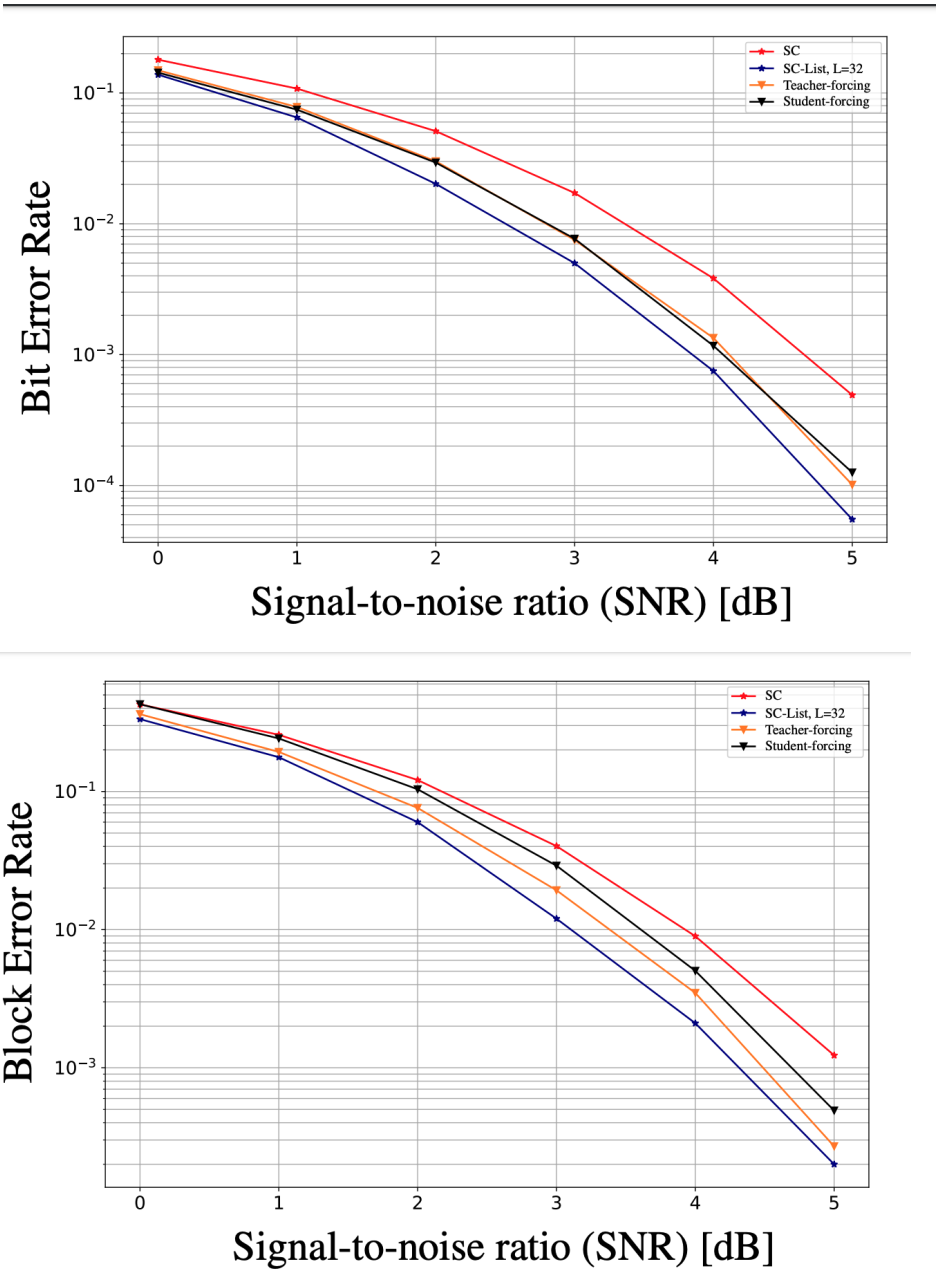


Figure 5.9: Training CRISP using student forcing results in sub-optimal BLER.

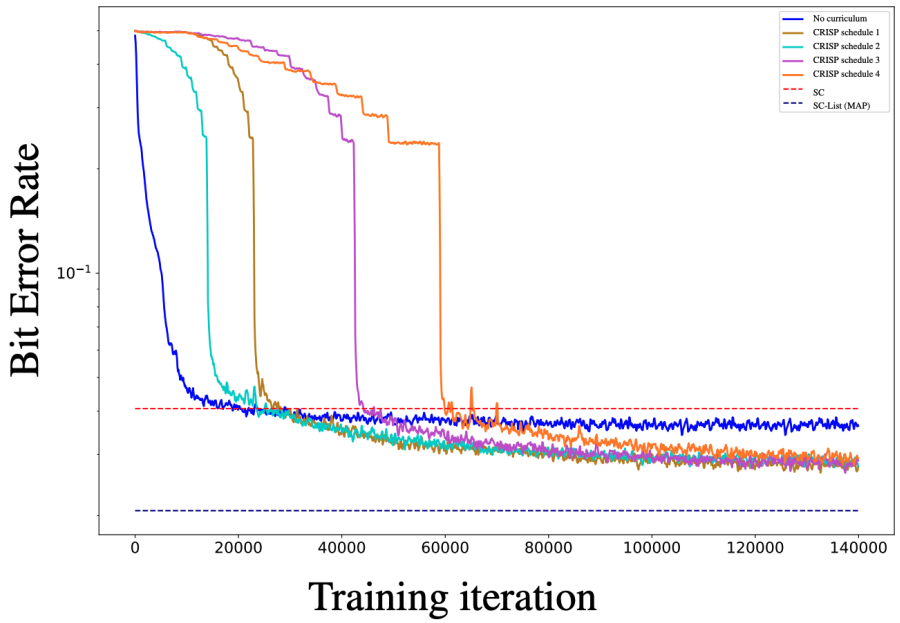


Figure 5.10: The number of iterations to train CRISP on each subcode using the curriculum are not critical to the final performance achieved.

CHAPTER 6

CONCLUSION

In this thesis, we propose machine-learning based channel decoders that outperform existing baselines. We use different machine learning paradigms, including model-based ML, deep reinforcement learning, and deep learning.

In Chapter 3 we presented TinyTurbo , a model-based ML approach to decode Turbo codes. TinyTurbo is an augmented version of the max-log-MAP algorithm with just 18 parameters learnt by using SGD to minimize an end-to-end loss function. We show that TinyTurbo approaches the performance of the MAP algorithm while having a complexity of the max-log-MAP algorithm. We also demonstrate robustness to various practical channels and demonstrate the generalizability of TinyTurbo to different blocklengths, rates, and trellises.

In Chapter 4, we presented a reinforcement learning-based method to decode short length PAC codes. We use the DQN algorithm to train an agent to search the PAC code tree to recover the transmitted message. In conjunction with a curriculum learning strategy, we show that DQN decoding achieves a reliability close to that of Fano decoding for PAC$(14, 32)$. However, this approach is not scalable to decode larger codes owing to its poor sample-complexity.

In Chapter 5, we described CRISP, a GRU-parametrized neural decoder to decode Polar and PAC codes, trained via curriculum learning. We show non-trivial gains over the baseline SC algorithm on Polar$(22, 64)$ and Polar$(16, 32)$. CRISP also shows a near-MAP performance on PAC$(16, 32)$, and is the first learning-based method to do so to the best of our knowledge. Due to the inherent parallelizability of neural networks, the DQN and CRISP methods achieve a better throughput than current methods.

Model-based machine learning methods promise scalable decoders that achieve good reliability with low computational complexity, as we demonstrate with TinyTurbo. However, in scenarios such as Polar decoding prior

work has shown that the above approach does not provide significant gains over the baseline SC algorithm. Deep learning architectures which exploit the structure of the encoder can achieve near-optimal results in channel decoding. However, this comes at a cost of computational complexity and difficulty to scale to large codes.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[2] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge University Press, 2008.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.

[4] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, p. 3051–3073, Jul 2009. [Online]. Available: http://dx.doi.org/10.1109/TIT.2009.2021379

[5] E. Arıkan, "From sequential decoding to channel polarization and back again," *arXiv preprint arXiv:1908.09594*, 2019.

[6] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[7] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Transactions on Information Theory*, vol. 9, no. 2, pp. 64–74, 1963.

[8] M. Rowshan, A. Burg, and E. Viterbo, "Complexity-efficient fano decoding of polarization-adjusted convolutional (pac) codes," in *2020 International Symposium on Information Theory and Its Applications (ISITA)*. IEEE, 2020, pp. 200–204.

[9] Z. Qin, H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep learning in physical layer communications," *IEEE Wireless Communications*, vol. 26, no. 2, pp. 93–99, 2019.

[10] H. Kim, S. Oh, and P. Viswanath, "Physical layer communication via deep learning," *IEEE Journal on Selected Areas in Information Theory*, 2020.

[11] N. Samuel, T. Diskin, and A. Wiesel, "Learning to detect," *IEEE Transactions on Signal Processing*, vol. 67, no. 10, pp. 2554–2564, 2019.

[12] S. Takabe, M. Imanishi, T. Wadayama, and K. Hayashi, "Deep learning-aided projected gradient detector for massive overloaded mimo channels," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[13] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "A model-driven deep learning network for mimo detection," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2018, pp. 584–588.

[14] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "Viterbinet: A deep learning based viterbi algorithm for symbol detection," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3319–3331, 2020.

[15] S. Khobahi, N. Shlezinger, M. Soltanalian, and Y. C. Eldar, "Lord-net: Unfolded deep detection network with low-resolution receivers," *IEEE Transactions on Signal Processing*, vol. 69, pp. 5651–5664, 2021.

[16] N. Farsad, N. Shlezinger, A. J. Goldsmith, and Y. C. Eldar, "Data-driven symbol detection via model-based machine learning," in *2021 IEEE Statistical Signal Processing Workshop (SSP)*. IEEE, 2021, pp. 571–575.

[17] H. Ye, G. Y. Li, and B.-H. Juang, "Power of deep learning for channel estimation and signal detection in ofdm systems," *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, 2017.

[18] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "Deep learning-based channel estimation for beamspace mmwave massive mimo systems," *IEEE Wireless Communications Letters*, vol. 7, no. 5, pp. 852–855, 2018.

[19] M. Soltani, V. Pourahmadi, A. Mirzaei, and H. Sheikhzadeh, "Deep learning-based channel estimation," *IEEE Communications Letters*, vol. 23, no. 4, pp. 652–655, 2019.

[20] T. Wang, C.-K. Wen, S. Jin, and G. Y. Li, "Deep learning-based csi feedback approach for time-varying massive mimo channels," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 416–419, 2018.

[21] J. Guo, C.-K. Wen, S. Jin, and G. Y. Li, "Convolutional neural network-based multiple-rate compressive sensing for massive mimo csi feedback: Design, simulation, and analysis," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2827–2840, 2020.

[22] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 341–346.

[23] T. O'shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[24] S. Dörner, S. Cammerer, J. Hoydis, and S. Ten Brink, "Deep learning based communication over the air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, 2017.

[25] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 1361–1365.

[26] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

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

[28] B. Vasić, X. Xiao, and S. Lin, "Learning to decode ldpc codes with finite-alphabet message passing," in *2018 Information Theory and Applications Workshop (ITA)*. IEEE, 2018, pp. 1–9.

[29] F. Liang, C. Shen, and F. Wu, "An iterative bp-cnn architecture for channel decoding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 144–159, 2018.

[30] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes-a syndrome-based approach," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1595–1599.

[31] C.-F. Teng, C.-H. D. Wu, A. K.-S. Ho, and A.-Y. A. Wu, "Low-complexity recurrent neural network-based polar decoder with weight quantization mechanism," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1413–1417.

[32] Y. Jiang, S. Kannan, H. Kim, S. Oh, H. Asnani, and P. Viswanath, "Deepturbo: Deep turbo decoder," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2019, pp. 1–5.

[33] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," *Advances in Neural Information Processing Systems*, vol. 32, pp. 2329–2339, 2019.

[34] A. Buchberger, C. Häger, H. D. Pfister, L. Schmalen, and A. G. Amat, "Pruning neural belief propagation decoders," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 338–342.

[35] Y. He, J. Zhang, S. Jin, C.-K. Wen, and G. Y. Li, "Model-driven dnn decoder for turbo codes: Design, simulation, and experimental results," *IEEE Transactions on Communications*, vol. 68, no. 10, pp. 6127–6140, 2020.

[36] S. Habib, A. Beemer, and J. Kliewer, "Learning to decode: Reinforcement learning for decoding of sparse graph-based channel codes," *arXiv preprint arXiv:2010.05637*, 2020.

[37] X. Chen and M. Ye, "Cyclically equivariant neural decoders for cyclic codes," *arXiv preprint arXiv:2105.05540*, 2021.

[38] X. Chen and M. Ye, "Improving the list decoding version of the cyclically equivariant neural decoder," *arXiv preprint arXiv:2106.07964*, 2021.

[39] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE, 2016, pp. 223–228.

[40] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," *Advances in neural information processing systems*, vol. 31, 2018.

[41] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels," in *Advances in Neural Information Processing Systems*, 2019, pp. 2758–2768.

[42] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Learn codes: Inventing low-latency codes via recurrent neural networks," *IEEE Journal on Selected Areas in Information Theory*, 2020.

[43] A. V. Makkuva, X. Liu, M. V. Jamali, H. Mahdavifar, S. Oh, and P. Viswanath, "Ko codes: inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7368–7378.

[44] M. V. Jamali, H. Saber, H. Hatami, and J. H. Bae, "Productae: Towards training larger channel codes based on neural product codes," *arXiv preprint arXiv:2110.04466*, 2021.

[45] K. Chahine, Y. Jiang, P. Nuti, H. Kim, and J. Cho, "Turbo autoencoder with a trainable interleaver," *arXiv preprint arXiv:2111.11410*, 2021.

[46] K. Chahine, N. Ye, and H. Kim, "Deepic: Coding for interference channels via deep learning," *arXiv preprint arXiv:2108.06028*, 2021.

[47] J. Clausius, S. Dörner, S. Cammerer, and S. ten Brink, "Serial vs. parallel turbo-autoencoders and accelerated training for learned channel codes," in *2021 11th International Symposium on Topics in Coding (ISTC)*. IEEE, 2021, pp. 1–5.

[48] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2017, pp. 1–6.

[49] T. K. Moon, *Error correction coding: mathematical methods and algorithms*. John Wiley & Sons, 2020.

[50] 3GPP, "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.212, 01 2011, version 10.0.0. [Online]. Available: https://www.3gpp.org/

[51] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on information theory*, vol. 20, no. 2, pp. 284–287, 1974.

[52] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge university press, 2009.

[53] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, 2013.

[54] M. Plotkin, "Binary codes with specified minimum distance," *IRE Transactions on Information Theory*, vol. 6, no. 4, pp. 445–450, 1960.

[55] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[56] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "Llr-based successive cancellation list decoding of polar codes," *IEEE transactions on signal processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[57] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast list decoders for polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 318–328, 2015.

[58] S. A. Hashemi, C. Condo, and W. J. Gross, "A fast polar code list decoder architecture based on sphere decoding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2368–2380, 2016.

[59] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "From polar to reed-muller codes: A technique to improve the finite-length performance," *IEEE Transactions on Communications*, vol. 62, no. 9, pp. 3084–3091, 2014.

[60] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE communications letters*, vol. 16, no. 12, pp. 2044–2047, 2012.

[61] K. Niu and K. Chen, "Crc-aided decoding of polar codes," *IEEE communications letters*, vol. 16, no. 10, pp. 1668–1671, 2012.

[62] V. Miloslavskaya and P. Trifonov, "Sequential decoding of polar codes," *IEEE Communications Letters*, vol. 18, no. 7, pp. 1127–1130, 2014.

[63] M. Moradi, A. Mozammel, K. Qin, and E. Arikan, "Performance and complexity of sequential decoding of pac codes," *arXiv preprint arXiv:2012.04990*, 2020.

[64] B. Li, H. Zhang, and J. Gu, "On pre-transformed polar codes," *arXiv preprint arXiv:1912.06359*, 2019.

[65] H. Yao, A. Fazeli, and A. Vardy, "List decoding of arıkan's pac codes," *Entropy*, vol. 23, no. 7, p. 841, 2021.

[66] R. Johannesson and K. S. Zigangirov, *Fundamentals of convolutional coding*. John Wiley & Sons, 2015.

[67] M. Rowshan, A. Burg, and E. Viterbo, "Polarization-adjusted convolutional (pac) codes: Fano decoding vs list decoding," *arXiv preprint arXiv:2002.06805*, 2020.

[68] M. Moradi, "On the metric and computation of pac codes," *arXiv preprint arXiv:2012.05511*, 2020.

[69] H. Zhu, Z. Cao, Y. Zhao, D. Li, and Y. Yang, "Fast list decoders for polarization-adjusted convolutional (pac) codes," *arXiv preprint arXiv:2012.09425*, 2020.

[70] M. Rowshan and E. Viterbo, "List viterbi decoding of pac codes," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 3, pp. 2428–2435, 2021.

[71] M. Rowshan and E. Viterbo, "On convolutional precoding in pac codes," in *2021 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2021, pp. 1–6.

[72] J.Vogt and A.Finger, "Improving the max-log-map turbo decoder," *Electron. Lett.,*, vol. 36, pp. 1937–1939, 2000.

[73] J. N. C. Chaikalis and F. Riera-Palou, "Improving the reconfigurable sova/log-map turbo decoder for 3gpp," *IEEE / IEE International Symposium on Communication Systems, Networks and DSP*, 2002.

[74] N. H. Yue DW, "Unified scaling factor approach for turbo decoding algorithms," *IET Commun.*, pp. 905–914, 2010.

[75] H. Claussen, H. Karimi, and B. Mulgrew, "Improved max-log map turbo decoding using maximum mutual information combining," in *14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications, 2003. PIMRC 2003.*, vol. 1, 2003, pp. 424–428 Vol.1.

[76] L. Sun and H. Wang, "Improved max-log-map turbo decoding by extrinsic information scaling and combining," in *Communications, Signal Processing, and Systems*, Q. Liang, X. Liu, Z. Na, W. Wang, J. Mu, and B. Zhang, Eds. Springer Singapore, 2020, pp. 336–344.

[77] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[78] Y. He, J. Zhang, S. Jin, C.-K. Wen, and G. Y. Li, "Model-driven dnn decoder for turbo codes: Design, simulation, and experimental results," *IEEE Transactions on Communications*, vol. 68, no. 10, pp. 6127–6140, 2020.

[79] M. Ebada, S. Cammerer, A. Elkelesh, and S. ten Brink, "Deep learning-based polar code design," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 177–183.

[80] A. V. Makkuva, X. Liu, M. V. Jamali, H. Mahdavifar, S. Oh, and P. Viswanath, "Ko codes: inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7368–7378.

[81] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception." 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.104, 04 2017, version 14.3.0. [Online]. Available: https://www.3gpp.org/

[82] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[83] 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.

[84] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[85] F. Carpi, C. Häger, M. Martalò, R. Raheli, and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, pp. 922–929.

[86] X. Wang, J. He, J. Li, and L. Shan, "Reinforcement learning for bit-flipping decoding of polar codes," *Entropy*, vol. 23, no. 2, p. 171, 2021.

[87] J. Gao and K. Niu, "A reinforcement learning based decoding method of short polar codes," in *2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2021, pp. 1–6.

[88] S. Habib, A. Beemer, and J. Kliewer, "Learned scheduling of ldpc decoders based on multi-armed bandits," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 2789–2794.

[89] S. Habib, A. Beemer, and J. Kliewer, "Reldec: Reinforcement learning-based decoding of moderate length ldpc codes," *arXiv preprint arXiv:2112.13934*, 2021.

[90] Y. Liao, S. A. Hashemi, J. M. Cioffi, and A. Goldsmith, "Construction of polar codes with reinforcement learning," *IEEE Transactions on Communications*, vol. 70, no. 1, pp. 185–198, 2021.

[91] N. Doan, S. A. Hashemi, and W. J. Gross, "Decoding polar codes with reinforcement learning," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[92] L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "Reinforcement learning for nested polar code construction," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[93] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[94] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[95] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[96] H. Sun, E. Viterbo, and R. Liu, "Optimized rate-profiling for pac codes," *arXiv preprint arXiv:2106.04074*, 2021.

[97] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," vol. abs/1412.3555, 2014.

[98] W. J. Gross, N. Doan, E. Ngomseu Mambou, and S. Ali Hashemi, "Deep learning techniques for decoding polar codes," *Machine learning for future wireless communications*, pp. 287–301, 2020.

[99] W. Lyu, Z. Zhang, C. Jiao, K. Qin, and H. Zhang, "Performance evaluation of channel decoding with deep neural networks," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[100] J. Seo, J. Lee, and K. Kim, "Decoding of polar code by using deep feed-forward neural networks," in *2018 international conference on computing, networking and communications (ICNC)*. IEEE, 2018, pp. 238–242.

[101] Z. Cao, H. Zhu, Y. Zhao, and D. Li, "Learning to denoise and decode: A novel residual neural network decoder for polar codes," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*. IEEE, 2020, pp. 1–6.

[102] W. Xu, Z. Wu, Y.-L. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *2017 IEEE International workshop on signal processing systems (SiPS)*. IEEE, 2017, pp. 1–6.

[103] W. Xu, X. You, C. Zhang, and Y. Be'ery, "Polar decoding on sparse graphs with deep learning," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2018, pp. 599–603.

[104] N. Doan, S. A. Hashemi, E. N. Mambou, T. Tonnellier, and W. J. Gross, "Neural belief propagation decoding of crc-polar concatenated codes," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[105] X. Wang, H. Zhang, R. Li, L. Huang, S. Dai, Y. Huangfu, and J. Wang, "Learning to flip successive cancellation decoding of polar codes with lstm networks," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2019, pp. 1–5.

[106] N. Doan, S. A. Hashemi, F. Ercan, and W. J. Gross, "Fast sc-flip decoding of polar codes with reinforcement learning," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[107] S. Cammerer, T. Gruber, J. Hoydis, and S. Ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *GLOBECOM 2017-2017 IEEE global communications conference*. IEEE, 2017, pp. 1–6.

[108] N. Doan, S. A. Hashemi, and W. J. Gross, "Neural successive cancellation decoding of polar codes," in *2018 IEEE 19th international workshop on signal processing advances in wireless communications (SPAWC)*. IEEE, 2018, pp. 1–5.

[109] H. Lee, E. Y. Seo, H. Ju, and S.-H. Kim, "On training neural network decoders of rate compatible polar codes via transfer learning," *Entropy*, vol. 22, no. 5, p. 496, 2020.

[110] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," 2016. [Online]. Available: https://arxiv.org/abs/1610.09038