
KO codes: Inventing Nonlinear Encoding and Decoding for Reliable Wireless Communication via Deep-learning

Ashok Vardhan Makkuva^{*1} Xiyang Liu^{*2} Mohammad Vahid Jamali³ Hessam MahdaviFar³ Sewoong Oh²
Pramod Viswanath¹

Abstract

Landmark codes underpin reliable physical layer communication, e.g., Reed-Muller, BCH, Convolution, Turbo, LDPC and Polar codes: each is a linear code and represents a mathematical breakthrough. The impact on humanity is huge: each of these codes has been used in global wireless communication standards (satellite, WiFi, cellular). Reliability of communication over the classical additive white Gaussian noise (AWGN) channel enables benchmarking and ranking of the different codes. In this paper, we construct KO codes, a computationally efficient family of deep-learning driven (encoder, decoder) pairs that outperform the state-of-the-art reliability performance on the standardized AWGN channel. KO codes beat state-of-the-art Reed-Muller and Polar codes, under the low-complexity successive cancellation decoding, in the challenging short-to-medium block length regime on the AWGN channel. We show that the gains of KO codes are primarily due to the nonlinear mapping of information bits directly to transmit real symbols (bypassing modulation) and yet possess an efficient, high performance decoder. The key technical innovation that renders this possible is design of a novel family of neural architectures inspired by the computation tree of the Kronecker Operation (KO) central to Reed-Muller and Polar codes. These architectures pave way for the discovery of a much richer class of hitherto unexplored nonlinear algebraic structures. The code is available at <https://github.com/deepcomm/KOcodes>.

1. Introduction

Physical layer communication underpins the information age (WiFi, cellular, cable and satellite modems). Codes, composed of encoder and decoder pairs, are the basic mathematical objects enabling reliable communication: encoder maps original data bits into a longer sequence, and decoders map the received sequence to the original bits. Reliability is precisely measured: bit error rate (BER) measures the fraction of input bits that were incorrectly decoded; block error rate (BLER) measures the fraction of times at least one of the original data bits was incorrectly decoded.

Landmark codes include Reed-Muller (RM), BCH, Turbo, LDPC and Polar codes (Richardson & Urbanke, 2008): each is a linear code and represents a mathematical breakthrough discovered over a span of six decades. The impact on humanity is huge: each of these codes has been used in global communication standards over the past six decades. These codes essentially operate at the information-theoretic limits of reliability over the additive white Gaussian noise (AWGN) channel, when the number of information bits is large, the so-called “large block length” regime. In the small and medium block length regimes, the state-of-the-art codes are *algebraic*: encoders and decoders are invented based on specific linear algebraic constructions over the binary and higher order fields and rings. Especially prominent binary algebraic codes are RM codes and closely related polar codes, whose encoders are recursively defined as Kronecker products of a simple linear operator and constitute the state of the art in small-to-medium block length regimes.

Inventing new codes is a major intellectual activity both in academia and the wireless industry; this is driven by emerging practical applications, e.g., low block length regime in Internet of Things (Ma et al., 2019). The core challenge is that the space of codes is very vast and the sizes astronomical; for instance a rate $1/2$ code over even 100 information bits involves designing 2^{100} codewords in a 200 dimensional space. Computationally efficient encoding and decoding procedures are a must, apart from high reliability. Thus, although a random code is information theoretically optimal, neither encoding nor decoding is computationally efficient. The mathematical landscape of computationally

^{*}Equal contribution ¹Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign ²Paul G. Allen School of Computer Science & Engineering, University of Washington ³Department of Electrical Engineering and Computer Science, University of Michigan. Correspondence to: Ashok, Xiyang <makkuva2@illinois.edu, xiyangl@cs.washington.edu>.

efficient codes has been plumbed over the decades by some of the finest mathematical minds, resulting in two distinct families of codes: *algebraic codes* (RM, BCH – focused on properties of polynomials) and *graph codes* (Turbo, LDPC – based on sparse graphs and statistical physics). The former is deterministic and involves discrete mathematics, while the latter harnesses randomness, graphs, and statistical physics to behave like a pseudorandom code. A major open question is the invention of new codes, and especially fascinating would be a family of codes outside of these two classes.

Our major result is the invention of a new family of codes, called KO codes, that have features of both code families: they are nonlinear generalizations of the Kronecker operation underlying the algebraic codes (e.g., Reed-Muller) parameterized by neural networks; the parameters are learnt in an end-to-end training paradigm in a data driven manner. Deep learning (DL) has transformed several domains of human endeavor that have traditionally relied heavily on mathematical ingenuity, e.g., game playing (AlphaZero (Silver et al., 2018)), biology (AlphaFold (Senior et al., 2019)), and physics (new laws (Udrescu & Tegmark, 2020)). Our results can be viewed as an added domain to the successes of DL in inventing mathematical structures.

A linear encoder is defined by a *generator matrix*, which maps information bits to a codeword. The RM and the Polar families construct their generator matrices by recursively applying the Kronecker product operation to a simple two-by-two matrix and then selecting rows from the resulting matrix. The careful choice in selecting these rows is driven by the desired algebraic structure of the code, which is central to achieving the large *minimum* pairwise distance between two codewords, a hallmark of the algebraic family. This encoder can be alternatively represented by a computation graph. The recursive Kronecker product corresponds to a complete binary tree, and row-selection corresponds to freezing a set of leaves in the tree, which we refer to as a “Plotkin tree”, inspired by the pioneering construction in (Plotkin, 1960).

The Plotkin tree skeleton allows us to tailor a new neural network architecture: we expand the algebraic family of codes by replacing the (linear) Plotkin construction with a non-linear operation parametrized by neural networks. The parameters are discovered by training the encoder with a matching decoder, that has the matching Plotkin tree as a skeleton, to minimize the error rate over (the unlimited) samples generated on AWGN channels.

Algebraic and the original RM codes promise a large worst-case pairwise distance (Alon et al., 2005). This ensures that RM codes achieve capacity in the large block length limit (Kudekar et al., 2017). However, for short block lengths, they are too conservative as we are interested in the average-case reliability. This is the gap KO codes exploit: we seek a

better average-case reliability and not the minimum pairwise distance.

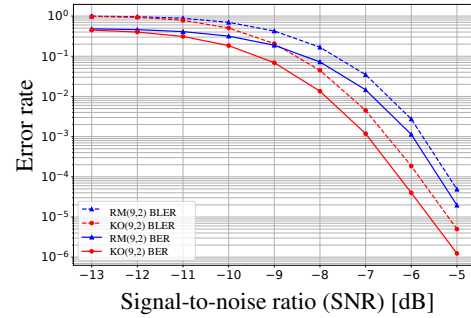


Figure 1. KO(9, 2), discovered by training a neural network with a carefully chosen architecture in §3, significantly improves upon state-of-the-art RM(9, 2) both in BER and BLER. (For both codes, the code block length is $2^9 = 512$ and the number of transmitted message bits is $\binom{9}{0} + \binom{9}{1} + \binom{9}{2} = 55$. Also, both codes are decoded using successive cancellation decoding with similar decoding complexity)

Figure 1 illustrates the gain for the example of RM(9, 2) code. Using the Plotkin tree of RM(9, 2) code as a skeleton, we design the KO(9, 2) code architecture and train on samples simulated over an AWGN channel. We discover a novel non-linear code and a corresponding efficient decoder that improves significantly over the RM(9, 2) code baseline, assuming both codes are decoded using successive cancellation decoding with similar decoding complexity. Analyzing the pairwise distances between two codewords reveals a surprising fact. The histogram for KO code nearly matches that of a random Gaussian codebook. The skeleton of the architecture from an algebraic family of codes, the training process with a variation of the stochastic gradient descent, and the simulated AWGN channel have worked together to discover a novel family of codes that harness the benefits of both algebraic and pseudorandom constructions.

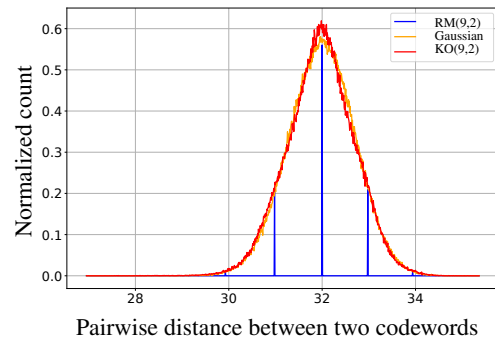


Figure 2. Histogram of pairwise distances between codewords of the KO(9, 2) code shows a strong resemblance to that of the Gaussian codebook, unlike the classical Reed-Muller code RM(9, 2).

In summary, we make the following contributions: We introduce novel neural network architectures for the (encoder, decoder) pair that generalizes the Kronecker operation central to RM/Polar codes. We propose training methods that discover novel non-linear codes when trained over AWGN and provide empirical results showing that this family of non-linear codes improves significantly upon the baseline code it was built on (both RM and Polar codes) whilst having the same encoding and decoding complexity. Interpreting the pairwise distances of the discovered codewords reveals that a KO code mimics the distribution of codewords from the random Gaussian codebook, which is known to be reliable but computationally challenging to decode. The decoding complexities of KO codes are $O(n \log n)$ where n is the block length, matching that of efficient decoders for RM and Polar codes.

We highlight that the design principle of KO codes serves as a general recipe to discover new family of non-linear codes improving upon their linear counterparts. In particular, the construction is not restricted to a specific decoding algorithm, such as successive cancellation (SC). In this paper, we focus on the SC decoding algorithm since it is one of the most efficient decoders for the RM and Polar family. At this decoding complexity, i.e. $O(n \log n)$, our results demonstrate that we achieve significant gain over these codes. Our preliminary results show that KO codes achieve similar gains over the RM codes, when both are decoded with list-decoding. We refer to §B for more details. Designing KO-inspired codes to improve upon the RPA decoder for RM codes (with complexity $O(n^r \log n)$ (Ye & Abbe, 2020)), and the list-decoded Polar codes (with complexity $O(Ln \log n)$ (Tal & Vardy, 2015)) where L is the list size, are promising active research directions, and outside the scope of this paper.

2. Problem formulation and background

We formally define the channel coding problem and provide background on Reed-Muller codes, the inspiration for our approach. Our notation is the following. We denote Euclidean vectors by bold face letters like \mathbf{m} , \mathbf{L} , etc. For $\mathbf{L} \in \mathbb{R}^n$, $\mathbf{L}_{k:m} \triangleq (L_k, \dots, L_m)$. If $\mathbf{v} \in \{0, 1\}^n$, we define the operator $\oplus_{\mathbf{v}}$ as $\mathbf{x} \oplus_{\mathbf{v}} \mathbf{y} \triangleq \mathbf{x} + (-1)^{\mathbf{v}} \mathbf{y}$.

2.1. Channel coding

Let $\mathbf{m} = (m_1, \dots, m_k) \in \{0, 1\}^k$ denote a block of *information/message bits* that we want to transmit. An encoder $g_{\theta}(\cdot)$ is a function parametrized by θ that maps these information bits into a binary vector \mathbf{x} of length n , i.e. $\mathbf{x} = g_{\theta}(\mathbf{m}) \in \{0, 1\}^n$. The *rate* $\rho = k/n$ of such a code measures how many bits of information we are sending per channel use. These codewords are transformed into real (or complex) valued signals, called modulation, before be-

ing transmitted over a channel. For example, Binary Phase Shift Keying (BPSK) modulation maps each $x_i \in \{0, 1\}$ to $1 - 2x_i \in \{\pm 1\}$ up to a universal scaling constant for all $i \in [n]$. Here, we do not strictly separate encoding from modulation and refer to both binary encoded symbols and real-valued transmitted symbols as *codewords*. The codewords also satisfy either a hard or soft power constraint. Here we consider the hard power constraint, i.e., $\|\mathbf{x}\|^2 = n$.

Upon transmission of this codeword \mathbf{x} across a noisy channel $P_{Y|X}(\cdot|\cdot)$, we receive its corrupted version $\mathbf{y} \in \mathbb{R}^n$. The decoder $f_{\phi}(\cdot)$ is a function parametrized by ϕ that subsequently processes the received vector \mathbf{y} to estimate the information bits $\hat{\mathbf{m}} = f_{\phi}(\mathbf{y})$. The closer $\hat{\mathbf{m}}$ is to \mathbf{m} , the more reliable the transmission. An error metric, such as Bit-Error-Rate (BER) or Block-Error-Rate (BLER), gauges the performance of the encoder-decoder pair (g_{θ}, f_{ϕ}) . Note that BER is defined as $\text{BER} \triangleq (1/k) \sum_i \mathbb{P}[\hat{m}_i \neq m_i]$, whereas $\text{BLER} \triangleq \mathbb{P}[\hat{\mathbf{m}} \neq \mathbf{m}]$.

The design of good codes given a channel and a fixed set of code parameters (k, n) can be formulated as:

$$(\theta, \phi) \in \arg \min_{\theta, \phi} \text{BER}(g_{\theta}, f_{\phi}), \quad (1)$$

which is a joint classification problem for k binary classes, and we train on the surrogate loss of cross entropy to make the objective differentiable. While classical optimal codes such as Turbo, LDPC, and Polar codes all have *linear* encoders, appropriately parametrizing both the encoder $g_{\theta}(\cdot)$ and the decoder $f_{\phi}(\cdot)$ by neural networks (NN) allows for a much broader class of codes, especially non-linear codes. However, in the absence of any structure, NNs fail to learn non-trivial codes and end up performing worse than simply repeating each message bit n/k times (Kim et al., 2018; Jiang et al., 2019b).

A fundamental question in machine learning for channel coding is thus: how do we design architectures for our neural encoders and decoders that give the appropriate inductive bias? To gain intuition towards addressing this, we focus on Reed-Muller (RM) codes. In §3, we present a novel family of non-linear codes, *KO codes*, that strictly generalize and improve upon RM codes by capitalizing on their inherent recursive structure. Our approach seamlessly generalizes to Polar codes, explained in §5.

2.2. Reed-Muller (RM) codes

We use a small example of RM(3, 1) and refer to Appendix E for the larger example in our main results.

Encoding. RM codes are a family of codes parametrized by a variable size $m \in \mathbb{Z}_+$ and an order $r \in \mathbb{Z}_+$ with $r \leq m$, denoted as RM(m, r). It is defined by an *encoder*, which maps binary information bits $\mathbf{m} \in \{0, 1\}^k$ to codewords $\mathbf{x} \in \{0, 1\}^n$. RM(m, r) code sends $k = \sum_{i=0}^r \binom{m}{i}$

information bits with $n = 2^m$ transmissions. The *code distance* measures the minimum distance between all (pairs of) codewords. Table 1 summarizes these parameters.

| Code length | Code dimension | Rate | Distance |
|-------------|---------------------------------|--------------|---------------|
| $n = 2^m$ | $k = \sum_{i=0}^r \binom{m}{i}$ | $\rho = k/n$ | $d = 2^{m-r}$ |

Table 1. Parameters of a RM(m, r) code

One way to define RM(m, r) code is via the recursive application of a *Plotkin construction*. The basic building block is a mapping Plotkin : $\{0, 1\}^\ell \times \{0, 1\}^\ell \rightarrow \{0, 1\}^{2\ell}$, where

$$\text{Plotkin}(\mathbf{u}, \mathbf{v}) = (\mathbf{u}, \mathbf{u} \oplus \mathbf{v}), \quad (2)$$

with \oplus representing a coordinate-wise XOR and (\cdot, \cdot) denoting concatenation of two vectors (Plotkin, 1960).

In view of the Plotkin construction, RM codes are recursively defined as a set of codewords of the form:

$$\text{RM}(m, r) = \{(\mathbf{u}, \mathbf{u} \oplus \mathbf{v}) : \mathbf{u} \in \text{RM}(m-1, r), \mathbf{v} \in \text{RM}(m-1, r-1)\}, \quad (3)$$

where RM($m, 0$) is a repetition code that repeats a single information bit 2^m times, i.e., $\mathbf{x} = (m_1, m_1, \dots, m_1)$. When $r = m$, the full-rate RM(m, m) code is also recursively defined as a Plotkin construction of two RM($m-1, m-1$) codes. Unrolling the recursion in Eq. (3), a RM(m, r) encoder can be represented by a corresponding (rooted and binary) computation tree, which we refer to as its *Plotkin tree*. In this tree, each branch represents a Plotkin mapping of two codes of appropriate lengths, recursively applied from the leaves to the root.

Figure 3a illustrates such a Plotkin tree decomposition of RM(3, 1) encoder. Encoding starts from the bottom right leaves. The leaf RM(1, 0) maps m_3 to (m_3, m_3) (repetition), and another leaf RM(1, 1) maps (m_1, m_2) to $(m_1, m_1 \oplus m_2)$ (Plotkin mapping of two RM(0, 0) codes). Each branch in this tree performs the Plotkin construction of Eq. (2). The next operation is the parent of these two leaves, which performs Plotkin(RM(1, 1), RM(1, 0)) = Plotkin($(m_1, m_1 \oplus m_2), (m_3, m_3)$) which outputs the vector $(m_1, m_1 \oplus m_2, m_1 \oplus m_3, m_1 \oplus m_2 \oplus m_3)$, which is known as RM(2, 1) code. This coordinate-wise Plotkin construction is applied recursively one more time to combine RM(2, 0) and RM(2, 1) at the root of the tree. The resulting codewords are RM(3, 1) = Plotkin(RM(2, 1), RM(2, 0)) = Plotkin($(m_1, m_1 \oplus m_2, m_1 \oplus m_3, m_1 \oplus m_2 \oplus m_3), (m_4, m_4, m_4, m_4)$).

This recursive structure of RM codes (i) inherits the good minimum distance property of the Plotkin construction and (ii) enables efficient decoding.

Decoding. Since (Reed, 1954), there have been several decoders for RM codes; (Abbe et al., 2020) is a detailed survey.

We focus on the most efficient one, called *Dumer's recursive decoding* (Dumer, 2004; 2006; Dumer & Shabunov, 2006b) that fully capitalizes on the recursive Plotkin construction in Eq. (3). The basic principle is: to decode an RM codeword $\mathbf{x} = (\mathbf{u}, \mathbf{u} \oplus \mathbf{v}) \in \text{RM}(m, r)$, we first recursively decode the left sub-codeword $\mathbf{v} \in \text{RM}(m-1, r-1)$ and then the right sub-codeword $\mathbf{u} \in \text{RM}(m-1, r)$, and we use them together to stitch back the original codeword. This recursion is continued until we reach the leaf nodes, where we perform maximum a posteriori (MAP) decoding. Dumer's recursive decoding is also referred to as *successive cancellation* decoding in the context of polar codes (Arikan, 2009).

Figure 3c illustrates this decoding procedure for RM(3, 1). Dumer's decoding starts at the root and uses the soft-information of codewords to decode the message bits. Suppose that the message bits $\mathbf{m} = (m_1, \dots, m_4)$ are encoded into an RM(3, 1) codeword $\mathbf{x} \in \{0, 1\}^8$ using the Plotkin encoder in Figure 3a. Let $\mathbf{y} \in \mathbb{R}^8$ be the corresponding noisy codeword received at the decoder. To decode the bits \mathbf{m} , we first obtain the soft-information of the codeword \mathbf{x} , i.e., we compute its Log-Likelihood-Ratio (LLR) $\mathbf{L} \in \mathbb{R}^8$:

$$L_i = \log \frac{\mathbb{P}[y_i | x_i = 0]}{\mathbb{P}[y_i | x_i = 1]}, \quad i = 1, \dots, 8.$$

We next use \mathbf{L} to compute soft-information for its left and right children: the RM(2, 0) codeword \mathbf{v} and the RM(2, 1) codeword \mathbf{u} . We start with the left child \mathbf{v} .

Since the codeword $\mathbf{x} = (\mathbf{u}, \mathbf{u} \oplus \mathbf{v})$, we can also represent its left child as $\mathbf{v} = \mathbf{u} \oplus (\mathbf{u} \oplus \mathbf{v}) = \mathbf{x}_{1:4} \oplus \mathbf{x}_{5:8}$. Hence its LLR vector $\mathbf{L}_v \in \mathbb{R}^4$ can be readily obtained from that of \mathbf{x} . In particular it is given by the log-sum-exponential transformation: $\mathbf{L}_v = \text{LSE}(\mathbf{L}_{1:4}, \mathbf{L}_{5:8})$, where $\text{LSE}(a, b) \triangleq \log((1+e^{a+b})/(e^a+e^b))$ for $a, b \in \mathbb{R}$. Since this feature \mathbf{L}_v corresponds to a repetition code, $\mathbf{v} = (m_4, m_4, m_4, m_4)$, majority decoding (same as the MAP) on the sign of \mathbf{L}_v yields the decoded message bit as \hat{m}_4 . Finally, the left codeword is decoded as $\hat{\mathbf{v}} = (\hat{m}_4, \hat{m}_4, \hat{m}_4, \hat{m}_4)$.

Having decoded the left RM(2, 0) codeword $\hat{\mathbf{v}}$, our goal is to now obtain soft-information $\mathbf{L}_u \in \mathbb{R}^4$ for the right RM(2, 1) codeword \mathbf{u} . Fixing $\mathbf{v} = \hat{\mathbf{v}}$, notice that the codeword $\mathbf{x} = (\mathbf{u}, \mathbf{u} \oplus \hat{\mathbf{v}})$ can be viewed as a 2-repetition of \mathbf{u} depending on the parity of $\hat{\mathbf{v}}$. Thus the LLR \mathbf{L}_u is given by LLR addition accounting for the parity of $\hat{\mathbf{v}}$: $\mathbf{L}_u = \mathbf{L}_{1:4} \oplus_{\hat{\mathbf{v}}} \mathbf{L}_{5:8} = \mathbf{L}_{1:4} + (-1)^{\hat{\mathbf{v}}} \mathbf{L}_{5:8}$. Since RM(2, 1) is an internal node in the tree, we again recursively decode its left child RM(1, 0) and its right child RM(1, 1), which are both leaves. For RM(1, 0), decoding is similar to that of RM(2, 0) above, and we obtain its information bit \hat{m}_3 by first applying the log-sum-exponential function on the feature \mathbf{L}_u and then majority decoding. Likewise, we obtain the LLR feature $\mathbf{L}_{uu} \in \mathbb{R}^2$ for the right RM(1, 1) child using parity-adjusted LLR addition on \mathbf{L}_u . Finally,

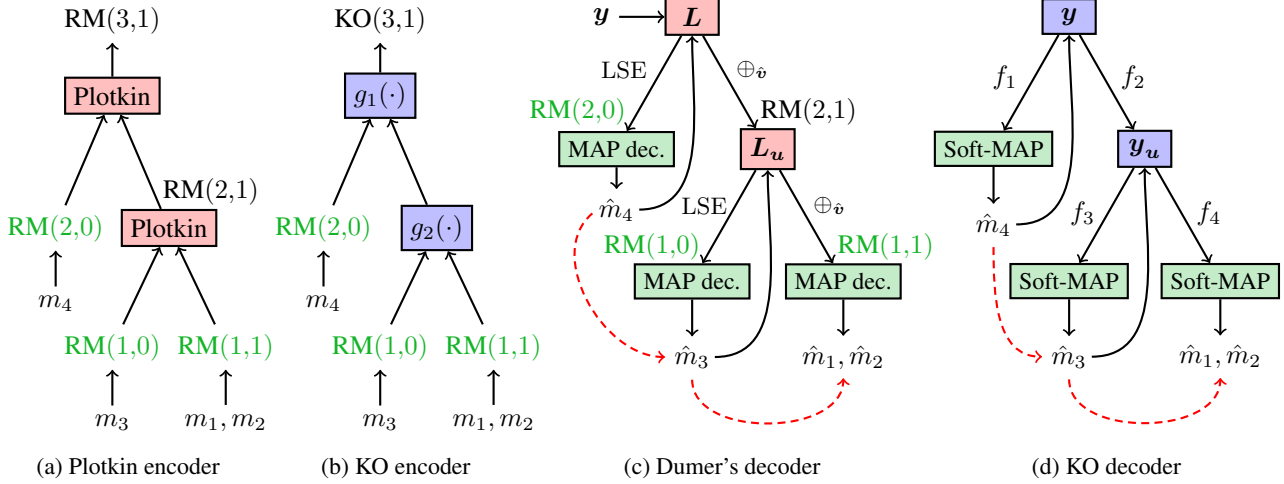


Figure 3. Plotkin trees for RM(3, 1) and KO(3, 1) codes; Leaves are shown in green. Red arrows indicate the bit decoding order.

we decode its corresponding bits (\hat{m}_1, \hat{m}_2) using efficient MAP-decoding of first order RM codes (Abbe et al., 2020). Thus we obtain the full block of decoded message bits as $\hat{\mathbf{m}} = (\hat{m}_1, \hat{m}_2, \hat{m}_3, \hat{m}_4)$.

An important observation from Dumer’s algorithm is that the sequence of bit decoding in the tree is: RM(2, 0) \rightarrow RM(1, 0) \rightarrow RM(1, 1). A similar decoding order holds for all RM(m , 2) codes, where all the left leaves (order-1 codes) are decoded first from top to bottom, and the right-most leaf (full-rate RM(2, 2)) is decoded at the end.

3. KO codes: Novel Neural codes

We design KO codes using the Plotkin tree as the skeleton of a new neural network architecture, which strictly improve upon their classical counterparts.

KO encoder. Earlier we saw the design of RM codes via recursive Plotkin mapping. Inspired by this elegant construction, we present a new family of codes, called *KO codes*, denoted as $\text{KO}(m, r, g_\theta, f_\phi)$. These codes are parametrized by a set of four parameters: a non-negative integer pair (m, r) , a finite set of encoder neural networks g_θ , and a finite set of decoder neural networks f_ϕ . In particular, for any fixed pair (m, r) , our KO encoder inherits the same code parameters (k, n, ρ) and the same Plotkin tree skeleton of the RM encoder. However, a critical distinguishing component of our $\text{KO}(m, r)$ encoder is a set of encoding neural networks $g_\theta = \{g_i\}$ that strictly generalize the Plotkin mapping: to each internal node i of the Plotkin tree, we associate a neural network g_i that applies a coordinate-wise real valued non-linear mapping $(\mathbf{u}, \mathbf{v}) \mapsto g_i(\mathbf{u}, \mathbf{v}) \in \mathbb{R}^{2\ell}$ as opposed to the classical binary valued Plotkin mapping $(\mathbf{u}, \mathbf{v}) \mapsto (\mathbf{u}, \mathbf{u} \oplus \mathbf{v}) \in \{0, 1\}^{2\ell}$. Figure 3b illustrates this for the KO(3, 1) encoder.

The significance of our KO encoder g_θ is that by allowing for general nonlinearities g_i to be learnt at each node we enable for a much richer and broader class of nonlinear encoders and codes to be discovered on a whole, which contribute to non-trivial gains over standard RM codes. Further, we have the same encoding complexity as that of an RM encoder since each $g_i : \mathbb{R}^2 \rightarrow \mathbb{R}$ is applied coordinate-wise on its vector inputs. The parameters of these neural networks g_i are trained via stochastic gradient descent on the cross entropy loss. See §G for experimental details.

KO decoder. Training the encoder is possible only if we have a corresponding decoder. This necessitates the need for an efficient family of matching decoders. Inspired by the Dumer’s decoder, we present a new family of *KO decoders* that fully capitalize on the recursive structure of KO encoders via the Plotkin tree.

Our KO decoder has three distinct features: (i) Neural decoder: The KO decoder architecture is parametrized by a set of decoding neural networks $f_\phi = \{(f_{2i-1}, f_{2i})\}$. Specifically, to each internal node i in the tree, we associate f_{2i-1} to its left branch whereas f_{2i} corresponds to the right branch. Figure 3d shows this for the KO(3, 1) decoder. The pair of decoding neural networks (f_{2i-1}, f_{2i}) can be viewed as matching decoders for the corresponding encoding network g_i : While g_i encodes the left and right codewords arriving at this node, the outputs of f_{2i-1} and f_{2i} represent appropriate Euclidean feature vectors for decoding them. Further, f_{2i-1} and f_{2i} can also be viewed as a generalization of Dumer’s decoding to nonlinear real codewords: f_{2i-1} generalizes the LSE function, while f_{2i} extends the operation $\oplus_{\hat{\mathbf{v}}}$. Note that both the functions f_{2i-1} and f_{2i} are also applied coordinate-wise and hence we inherit the same decoding complexity as Dumer’s. (ii) Soft-MAP decoding: Since the classical MAP decoding to decode the bits at the leaves is not dif-

ferentiable, we design a new differentiable counterpart, the *Soft-MAP decoder*. Soft-MAP decoder enables gradients to pass through it, which is crucial for training the neural (encoder, decoder) pair (g_θ, f_ϕ) in an end-to-end manner. (iii) Channel agnostic: Our decoder directly operates on the received noisy codeword $\mathbf{y} \in \mathbb{R}^n$ while Dumer’s decoder uses its LLR transformation $\mathbf{L} \in \mathbb{R}^n$. Thus, our decoder can learn the appropriate channel statistics for decoding directly from \mathbf{y} alone; in contrast, Dumer’s algorithm requires precise channel characterization, which is not usually known.

4. Main results

We train the KO encoder g_θ and KO decoder f_ϕ from §3 using an approximation of the BER loss in (1). The details are provided in §G. In this section we focus on the second-order KO(8, 2) and KO(9, 2) codes.

4.1. KO codes improve over RM codes

In Figure 1, the trained KO(9, 2) improves over the competing RM(9, 2) both in BER and BLER. The superiority in BLER is unexpected as our training loss is a surrogate for the BER. Though one would prefer to train on BLER as it is more relevant in practice, it is challenging to design a surrogate loss for BLER that is also differentiable: all literature on learning decoders minimize only BER (Kim et al., 2020; Nachmani et al., 2018; Dörner et al., 2017). Consequently, improvements in BLER with trained encoders and/or decoders are rare. We discover a code that improves both BER and BLER, and we observe a similar gain with KO(8, 2) in Figure 4. Performance of a binarized version KO-b(8, 2) is also shown, which we describe further in §4.4.

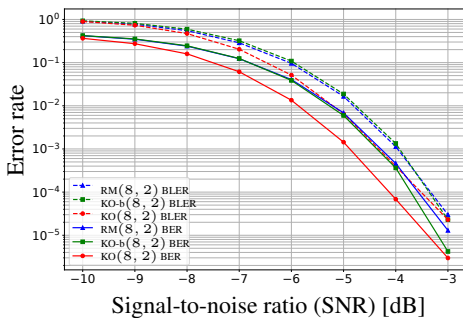


Figure 4. Neural network based KO(8, 2) and KO-b(8, 2) improve upon RM(8, 2) in BER and BLER, but the gain is small for the binarized codewords of KO-b(8, 2) (for all the codes, the code dimension is 37 and block length is 256).

4.2. Interpreting KO codes

We interpret the learned encoders and decoders to explain the source of the performance gain.

Interpreting the KO encoder. To interpret the learned KO code, we examine the pairwise distance between codewords. In classical linear coding, pairwise distances are expressed in terms of the weight distribution of the code, which counts how many codewords of each specific Hamming weight $1, 2, \dots, n$ exist in the code. The weight distribution of linear codes are used to derive analytical bounds, that can be explicitly computed, on the BER and BLER over AWGN channels (Sason & Shamai, 2006). For nonlinear codes, however, the weight distribution does not capture pairwise distances. Therefore, we explore the distribution of all the pairwise distances of non-linear KO codes that can play the same role as the weight distribution does for linear codes.

The pairwise distance distribution of the RM codes remains an active area of research as it is used to prove that RM codes achieve the capacity (Kaufman et al., 2012; Abbe et al., 2015; Sberlo & Shpilka, 2020) (Figure 5 blue). However, these results are asymptotic in the block length and do not guarantee a good performance, especially in the small-to-medium block lengths that we are interested in. On the other hand, Gaussian codebooks, codebooks randomly picked from the ensemble of all Gaussian codebooks, are known to be asymptotically optimal, i.e., achieving the capacity (Shannon, 1948), and also demonstrate optimal finite-length scaling laws closely related to the pairwise distance distribution (Polyanskiy et al., 2010) (Figure 5 orange).

Remarkably, the pairwise distance distribution of KO code shows a staggering resemblance to that of the Gaussian codebook of the same rate ρ and blocklength n (Figure 5 red). This is an unexpected phenomenon since we minimize only BER. We posit that the NN training has learned to construct a Gaussian-like codebook, in order to minimize BER. Most importantly, unlike the Gaussian codebook, KO codes constructed via NN training are fully compatible with efficient decoding. This phenomenon is observed for all order-2 codes we trained (e.g., Figure 2 for KO(9, 2)).

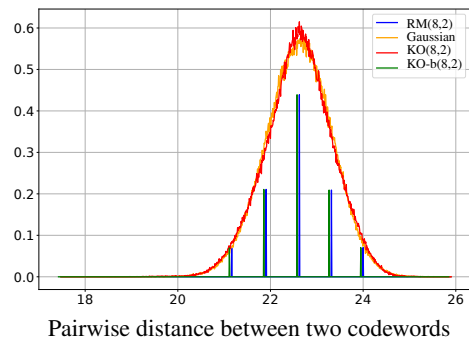


Figure 5. Histograms of pairwise distances between codewords for (8, 2) codes reveal that KO(8, 2) code has learned an approximate Gaussian codebook that can be efficiently decoded.

Interpreting the KO decoder. We now analyze how the KO decoder contributes to the gains in BLER over the RM decoder. Let $\mathbf{m} = (\mathbf{m}_{(7,1)}, \dots, \mathbf{m}_{(2,2)})$ denote the block of transmitted message bits, where the ordered set of indices $\mathcal{L} = \{(7,1), \dots, (2,2)\}$ correspond to the leaf branches (RM codes) of the Plotkin tree. Let $\hat{\mathbf{m}}$ be the decoded estimate by the KO(8,2) decoder.

We provide Plotkin trees of RM(8,2) and KO(8,2) decoders in Figures 15a and 15b in the appendix. Recall that for this KO(8,2) decoder, similar to the KO(3,1) decoder in Figure 3d, we decode each sub-code in the leaves sequentially, starting from the (7,1) branch down to (2,2): $\hat{\mathbf{m}}_{(7,1)} \rightarrow \dots \rightarrow \hat{\mathbf{m}}_{(2,2)}$. In view of this decoding order, BLER, defined as $\mathbb{P}[\hat{\mathbf{m}} \neq \mathbf{m}]$, can be decomposed as

$$\mathbb{P}[\hat{\mathbf{m}} \neq \mathbf{m}] = \sum_{i \in \mathcal{L}} \mathbb{P}[\hat{\mathbf{m}}_i \neq \mathbf{m}_i, \hat{\mathbf{m}}_{1:i-1} = \mathbf{m}_{1:i-1}]. \quad (4)$$

In other words, BLER can also be represented as the sum of the fraction of errors the decoder makes in each of the leaf branches when no errors were made in the previous ones. Thus, each term in Eq. (4) can be viewed as the contribution of each sub-code to the total BLER.

This is plotted in Figure 6, which shows that the KO(8,2) decoder achieves better BLER than the RM(8,2) decoder by making major gains in the leftmost (7,1) branch (which is decoded first) at the expense of other branches. However, the decoder (together with the encoder) has learnt to better balance these contributions evenly across all branches, resulting in lower BLER overall. The unequal errors in the branches of the RM code has been observed before, and some efforts made to balance them (Dumer & Shabunov, 2001); that KO codes learn such a balancing scheme purely from data is, perhaps, remarkable.

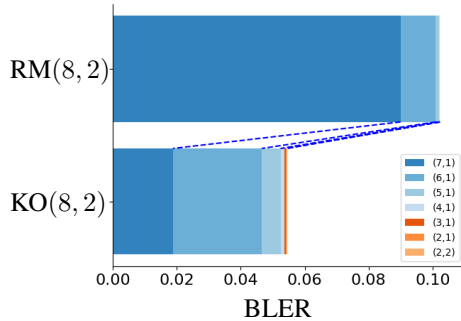


Figure 6. Separating each sub-code contribution in the KO(8,2) decoder and the RM(8,2) decoder reveals that KO(8,2) improves in the total BLER by balancing the contributions more evenly over the sub-codes.

4.3. Robustness to non-AWGN channels

As the environment changes dynamically in real world channels, robustness is crucial in practice. We therefore test the KO code under canonical channel models and demonstrate robustness, i.e., the ability of a code trained on AWGN to perform well under a different channel *without retraining*. It is well known that Gaussian noise is the worst case noise among all noise with the same variance (Lapidoth, 1996; Shannon, 1948) when an optimal decoder is used, which might take an exponential time. When decoded with efficient decoders, as we do with both RM and KO codes, catastrophic failures have been reported in the case of Turbo decoders (Kim et al., 2018). We show that both RM codes and KO codes are robust and that KO codes maintain their gains over RM codes as the channels vary.

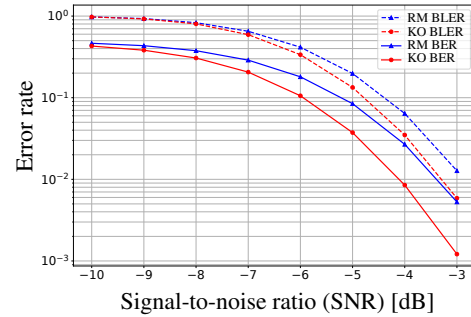


Figure 7. KO(8,2) trained on AWGN is robust when tested on a fast fading channel and maintains a significant gain over RM(8,2).

We first test on a *Rayleigh fast fading channel*, defined as $y_i = a_i x_i + n_i$, where x_i is the transmitted symbol, y_i is the received symbol, $n_i \sim \mathcal{N}(0, \sigma^2)$ is the additive Gaussian noise, and a is from a Rayleigh distribution with the variance of a chosen as $\mathbb{E}[a_i^2] = 1$.

We next test on a *bursty channel*, defined as $y_i = x_i + n_i + w_i$, where x_i is the input symbol, y_i is the received symbol, $n_i \sim \mathcal{N}(0, \sigma^2)$ is the additive Gaussian noise, and $w_i \sim \mathcal{N}(0, \sigma_b^2)$ with probability ρ and $w_i = 0$ with probability $1 - \rho$. In the experiment, we choose $\rho = 0.1$ and $\sigma_b = \sqrt{2}\sigma$.

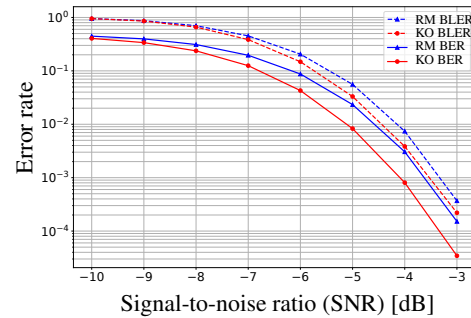


Figure 8. KO(8,2) trained on AWGN is robust when tested on a bursty channel and maintains a significant gain over RM(8,2).

4.4. Ablation studies

In comparison to the classical RM codes, the KO codes have two additional features: real-valued codewords and non-linearity. It is thus natural to ask how each of these components contribute to its gains over RM codes. To evaluate their contribution, we did ablation experiments for $\text{KO}(8, 2)$: (i) First, we constrain the KO codewords to be binary but allow for non-linearity in the encoder g_θ . The performance of this binarized version $\text{KO-b}(8, 2)$ is illustrated in Figure 9 below. We observe that this binarized $\text{KO-b}(8, 2)$ performs similar to $\text{RM}(8, 2)$ except for slight gains at high SNRs but uniformly worse than $\text{KO}(8, 2)$. (ii) Now we transmit the real-valued codewords but constrain the encoder g_θ to be linear (in real-value operations). The resulting code, $\text{KO-linear}(8, 2)$, performs almost identical to $\text{RM}(8, 2)$ but worse than $\text{KO}(8, 2)$, as highlighted by the orange curve in Figure 9.

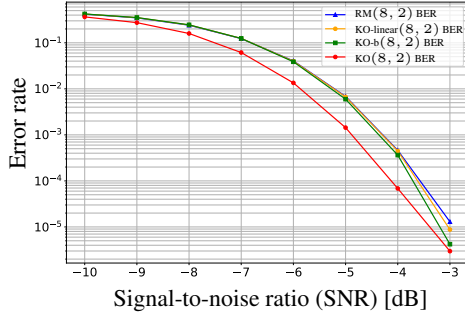


Figure 9. Ablation studies highlight that both non-linearity and real-valued codewords are equally important for good performance of KO codes. The linear version, $\text{KO-linear}(8, 2)$, and the binary version, $\text{KO-b}(8, 2)$, both perform worse than $\text{KO}(8, 2)$ and similar to $\text{RM}(8, 2)$.

These ablation experiments suggest us that presence of both the non-linearity and real-valued codewords are necessary for the good performance of KO codes and removal of any of these components hurts the gains it achieves over RM codes. Further, this also highlights that in absence of either of these components, the performance drops back to that of the original RM codes.

4.5. Complexity of KO decoding

Ultra-Reliable Low Latency Communication (URLLC) is increasingly required for modern applications including vehicular communication, virtual reality, and remote robotics (Sybis et al., 2016; Jiang et al., 2020). In general, a $\text{KO}(m, r)$ code requires $O(n \log n)$ operations to decode which is the same as the efficient Dumer’s decoder for an $\text{RM}(m, r)$ code, where $n = 2^m$ is the block length. More precisely, the successive cancellation decoder for

$\text{RM}(8, 2)$ requires 11268 operations whereas $\text{KO}(8, 2)$ requires 550644 operations which we did not try to optimize for this project. We discuss promising preliminary results in reducing the computational complexity in §6, where KO decoders achieve a computational efficiency comparable to the successive cancellation decoders of RM codes.

5. KO codes improve upon Polar codes

Results from §4 demonstrate that our KO codes significantly improve upon RM codes on a variety of benchmarks. Here, we focus on a different family of capacity-achieving landmark codes: *Polar codes* (Arikan, 2009).

Polar and RM codes are closely related, especially from an encoding point of view. The generator matrices of both codes are chosen from the same parent square matrix by following different row selection rules. More precisely, consider a $\text{RM}(m, r)$ code that has code dimension $k = \sum_{i=0}^r \binom{m}{i}$ and blocklength $n = 2^m$. Its encoding generator matrix is obtained by picking the k rows of the square matrix $\mathbf{G}_{n \times n} := \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{\otimes m}$ that have the largest Hamming weights (i.e., Hamming weight of at least 2^{m-r}), where $[\cdot]^{\otimes m}$ denotes the m -th Kronecker power. The Polar encoder, on the other hand, picks the rows of $\mathbf{G}_{n \times n}$ that correspond to the most reliable bit-channels (Arikan, 2009).

The recursive Kronecker structure inherent to the parent matrix $\mathbf{G}_{n \times n}$ can also be represented by a computation graph: a complete binary tree. Thus the corresponding computation tree for a Polar code is obtained by freezing a set of leaves (row-selection). We refer to this encoding computation graph of a Polar code as its *Plotkin tree*. This Plotkin tree structure of Polar codes enables a matching efficient decoder: the *successive cancellation* (SC). The SC decoding algorithm is similar to Dumer’s decoding for RM codes. Hence, Polar codes can be completely characterized by their corresponding Plotkin trees.

Inspired by the Kronecker structure of Polar Plotkin trees, we design a new family of KO codes to strictly improve upon them. We build a novel NN architecture that capitalizes on the Plotkin tree skeleton and generalizes it to nonlinear codes. This enables us to discover new nonlinear algebraic structures. The KO encoder and decoder can be trained in an end-to-end manner using variants of stochastic gradient descent (§A).

In Figure 10, we compare the performance of our KO code with its competing Polar(64, 7) code, i.e., code dimension $k = 7$ and block length $n = 64$, in terms of BER. Figure 10 highlights that our KO code achieves significant gains over Polar(64, 7) on a wide range of SNRs. In particular, we obtain a gain of almost 0.7 dB compared to that of Polar at the BER 10^{-4} . For comparison we also plot the per-

formance of both codes with the optimal MAP decoding. We observe that the BER curve of our KO decoder, unlike the SC decoder, almost matches that of the MAP decoder, convincingly demonstrating its optimality.

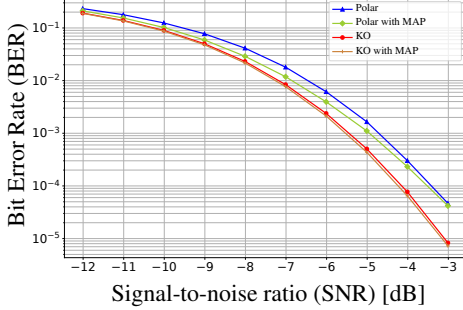


Figure 10. Neural network based KO code improves upon the Polar(64, 7) code when trained on AWGN channel. KO decoder also matches the optimal MAP decoder.

We also observe similar improvements for BLER (Figure 12, §A). This successful case study with training KO (encoder, decoder) pairs further demonstrates that our novel neural architectures seamlessly generalize to codes with an underlying Kronecker product structure.

6. Tiny KO

In this section we focus on further reducing the total number of mathematical operations required for our KO decoder with the objective of achieving similar computational efficiency as the successive cancellation decoder of RM codes.

As detailed in §G.3, each neural component in the KO encoder and decoder has 3 hidden layers with 32 nodes each. For the decoder, the total number of parameters in each decoder neural block is 69×32 . We replace all neural blocks with a smaller one with 1 hidden layer of 4 nodes. This decoder neural block has 20 parameters, obtaining a factor of 110 compression in the number of parameters. The computational complexity of this compressed decoder, which we refer to as TinyKO, is within a factor of 4 from Dumer’s successive cancellation decoder. Each neural network component has two matrix multiplication steps and one activation function on a vector, which can be fully parallelized on a GPU. With the GPU parallelization, TinyKO has the same time complexity/latency as Dumer’s SC decoding.

Table 2 shows that there is almost no loss in reliability for the compressed KO(8, 2) encoder and decoder in this manner. Training a smaller neural network take about two times more iterations compared to the larger one, although each iteration is faster for the smaller network.

If one is allowed more computation time (e.g., $O(n^r \log n)$),

| SNR (dB) | TinyKO(8, 2) BER | KO(8, 2) BER |
|----------|-----------------------|-----------------------|
| -10 | $0.38414 \pm 2e-7$ | $0.36555 \pm 2e-7$ |
| -9 | $0.29671 \pm 2e-7$ | $0.27428 \pm 2e-7$ |
| -8 | $0.18037 \pm 2e-7$ | $0.15890 \pm 2e-7$ |
| -7 | $0.07455 \pm 2e-7$ | $0.06167 \pm 1e-7$ |
| -6 | $0.01797 \pm 8e-8$ | $0.01349 \pm 7e-8$ |
| -5 | $2.18083e-3 \pm 3e-8$ | $1.46003e-3 \pm 2e-8$ |
| -4 | $1.18919e-4 \pm 7e-9$ | $0.64702e-4 \pm 4e-9$ |
| -3 | $4.54054e-6 \pm 1e-9$ | $3.16216e-6 \pm 1e-9$ |

Table 2. The smaller TinyKO neural architecture with 100 times smaller number of parameters achieve similar bit-error-rates as the bigger KO architecture.

then (Ye & Abbe, 2020) proposes a recursive projection-aggregation (RPA) decoder for $RM(m, r)$ codes that significantly improves over Dumer’s successive cancellation. With list decoding, this is empirically shown to approach the performance of the MAP decoder. It is a promising direction to explore deep learning architectures upon the computation tree of the RPA decoders to design new family of codes.

7. Related work

There is tremendous interest in the coding theory community to incorporate deep learning methods. In the context of channel coding, the bulk of the works focus on decoding known linear codes using data-driven neural decoders (Nachmani et al., 2016; O’shea & Hoydis, 2017; Dörner et al., 2017; Gruber et al., 2017; Nachmani et al., 2018; Kim et al., 2018; Vasić et al., 2018; Teng et al., 2019; Jiang et al., 2019a; Nachmani & Wolf, 2019; Buchberger et al., 2020; Habib et al., 2020; Chen & Ye, 2021); even here, most works have limited themselves to small block lengths due to the difficulty in generalization (for instance, even when nearly 90% of the codewords of a rate 1/2 Polar code over 8 information bits are exposed to the neural decoder (Gruber et al., 2017)).

On the other hand, very few works in the literature focus on discovering *both* encoders and decoders; the few which do, operate at very small block lengths (O’Shea et al., 2016; O’shea & Hoydis, 2017). One of the major challenges here is to jointly train the (encoder, decoder) pairs without getting stuck in local optima as the losses are non-convex. In (Jiang et al., 2019b), the authors employ clever training tricks to learn a novel autoencoder based codes that outperform the classical Turbo codes, which are sequential in nature. In contrast, here we focus on the generalizations of the Kronecker operation that underpins the RM and Polar family.

RM and Polar codes have seen active research, especially on improving decoding using neural networks: (Tallini & Cull, 1995; Xu et al., 2017; Cammerer et al., 2017; Ben-

natan et al., 2018; Doan et al., 2018; Lian et al., 2019; Wang et al., 2019; Carpi et al., 2019; Ebada et al., 2019). A common theme across majority of these works is to consider iterative/sequential decoding algorithms, such as belief propagation (BP), bit-flipping (BF), etc., and improve upon their performance by introducing learnable neural network components in them. On the other hand, our KO decoder strictly improves upon the natural SC decoder. Further, we learn the matching KO encoder whereas the encoding is fixed for them.

8. Conclusion

We introduce KO codes that generalize the recursive Kronecker operation crucial to designing RM and Polar codes. Using the computation tree (known as a Plotkin tree) of these classical codes as a skeleton, we propose a novel neural network architecture tailored for channel communication. Training over the AWGN channel, we discover the first family of *non-linear* codes that are not built upon any linear structure. KO codes significantly outperform the baseline Polar and RM codes under similar successive cancellation decoding architectures, which we call Dumer's decoder for the RM codes. The pairwise distance profile reveals that KO code combines the analytical structure of algebraic codes with the random structure of the celebrated random Gaussian codes.

Acknowledgements

Ashok would like to thank his colleagues Mona Zehni and Konik Kothari for helpful discussions about the project.

References

- Abbe, E., Shpilka, A., and Wigderson, A. Reed-muller codes for random erasures and errors. *IEEE Transactions on Information Theory*, 61(10):5229–5252, 2015.
- Abbe, E., Shpilka, A., and Ye, M. Reed-muller codes: Theory and algorithms, 2020.
- Alon, N., Kaufman, T., Krivelevich, M., Litsyn, S., and Ron, D. Testing reed-muller codes. *IEEE Transactions on Information Theory*, 51(11):4032–4039, 2005.
- Arikan, E. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.
- Bennatan, A., Choukroun, Y., and Kisilev, P. Deep learning for decoding of linear codes-a syndrome-based approach. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1595–1599. IEEE, 2018.
- Buchberger, A., Häger, C., Pfister, H. D., Schmalen, L., and Amat, A. G. Pruning neural belief propagation decoders. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pp. 338–342. IEEE, 2020.
- Cammerer, S., Gruber, T., Hoydis, J., and Ten Brink, S. Scaling deep learning-based decoding of polar codes via partitioning. In *GLOBECOM 2017-2017 IEEE global communications conference*, pp. 1–6. IEEE, 2017.
- Carpi, F., Häger, C., Martalò, M., Raheli, R., and Pfister, H. D. Reinforcement learning for channel coding: Learned bit-flipping decoding. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 922–929. IEEE, 2019.
- Chen, X. and Ye, M. Cyclically equivariant neural decoders for cyclic codes. *arXiv preprint arXiv:2105.05540*, 2021.
- Doan, N., Hashemi, S. A., and Gross, W. J. Neural successive cancellation decoding of polar codes. In *2018 IEEE 19th international workshop on signal processing advances in wireless communications (SPAWC)*, pp. 1–5. IEEE, 2018.
- Dörner, S., Cammerer, S., Hoydis, J., and Ten Brink, S. Deep learning based communication over the air. *IEEE Journal of Selected Topics in Signal Processing*, 12(1): 132–143, 2017.
- Dumer, I. Recursive decoding and its performance for low-rate reed-muller codes. *IEEE Transactions on Information Theory*, 50(5):811–823, 2004.
- Dumer, I. Soft-decision decoding of reed-muller codes: a simplified algorithm. *IEEE transactions on information theory*, 52(3):954–963, 2006.
- Dumer, I. and Shabunov, K. Near-optimum decoding for subcodes of reed-muller codes. In *Proceedings. 2001 IEEE International Symposium on Information Theory*, pp. 329. IEEE, 2001.
- Dumer, I. and Shabunov, K. Soft-decision decoding of reed-muller codes: recursive lists. *IEEE Transactions on information theory*, 52(3):1260–1266, 2006a.
- Dumer, I. and Shabunov, K. Soft-decision decoding of reed-muller codes: recursive lists. *IEEE Transactions on information theory*, 52(3):1260–1266, 2006b.
- Ebada, M., Cammerer, S., Elkelesh, A., and ten Brink, S. Deep learning-based polar code design. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 177–183. IEEE, 2019.
- Gruber, T., Cammerer, S., Hoydis, J., and ten Brink, S. On deep learning-based channel decoding. In *2017 51st*

- Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6. IEEE, 2017.
- Habib, S., Beemer, A., and Kliewer, J. Learning to decode: Reinforcement learning for decoding of sparse graph-based channel codes. *arXiv preprint arXiv:2010.05637*, 2020.
- Jamali, M. V., Liu, X., Makkuva, A. V., MahdaviFar, H., Oh, S., and Viswanath, P. Reed-Muller subcodes: Machine learning-aided design of efficient soft recursive decoding. *arXiv preprint arXiv:2102.01671*, 2021.
- Jiang, Y., Kannan, S., Kim, H., Oh, S., Asnani, H., and Viswanath, P. Deepturbo: Deep turbo decoder. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5. IEEE, 2019a.
- Jiang, Y., Kim, H., Asnani, H., Kannan, S., Oh, S., and Viswanath, P. Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels. In *Advances in Neural Information Processing Systems*, pp. 2758–2768, 2019b.
- Jiang, Y., Kim, H., Asnani, H., Kannan, S., Oh, S., and Viswanath, P. Learn codes: Inventing low-latency codes via recurrent neural networks. *IEEE Journal on Selected Areas in Information Theory*, 2020.
- Kaufman, T., Lovett, S., and Porat, E. Weight distribution and list-decoding size of reed-muller codes. *IEEE transactions on information theory*, 58(5):2689–2696, 2012.
- Kim, H., Jiang, Y., Rana, R., Kannan, S., Oh, S., and Viswanath, P. Communication algorithms via deep learning. *arXiv preprint arXiv:1805.09317*, 2018.
- Kim, H., Oh, S., and Viswanath, P. Physical layer communication via deep learning. *IEEE Journal on Selected Areas in Information Theory*, 2020.
- Kudekar, S., Kumar, S., Mondelli, M., Pfister, H. D., Şaşıoğlu, E., and Urbanke, R. L. Reed-muller codes achieve capacity on erasure channels. *IEEE Transactions on information theory*, 63(7):4298–4316, 2017.
- Lapidoth, A. Nearest neighbor decoding for additive non-gaussian noise channels. *IEEE Transactions on Information Theory*, 42(5):1520–1529, 1996.
- Lian, M., Carpi, F., Häger, C., and Pfister, H. D. Learned belief-propagation decoding with simple scaling and snr adaptation. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 161–165. IEEE, 2019.
- Ma, Z., Xiao, M., Xiao, Y., Pang, Z., Poor, H. V., and Vucetic, B. High-reliability and low-latency wireless communication for internet of things: challenges, fundamentals, and enabling technologies. *IEEE Internet of Things Journal*, 6(5):7946–7970, 2019.
- Nachmani, E. and Wolf, L. Hyper-graph-network decoders for block codes. *Advances in Neural Information Processing Systems*, 32:2329–2339, 2019.
- Nachmani, E., Be’ery, Y., and Burshtein, D. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346. IEEE, 2016.
- Nachmani, E., Marciano, E., Lugosch, L., Gross, W. J., Burshtein, D., and Be’ery, Y. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018.
- O’Shea, T. J., Karra, K., and Clancy, T. C. Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention. In *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 223–228. IEEE, 2016.
- O’shea, T. and Hoydis, J. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.
- Plotkin, M. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6(4):445–450, 1960.
- Polyanskiy, Y., Poor, H. V., and Verdú, S. Channel coding rate in the finite blocklength regime. *IEEE Transactions on Information Theory*, 56(5):2307–2359, 2010.
- Reed, I. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954.
- Richardson, T. and Urbanke, R. *Modern coding theory*. Cambridge University Press, 2008.
- Sason, I. and Shamai, S. Performance analysis of linear codes under maximum-likelihood decoding: A tutorial. 2006.
- Sberlo, O. and Shpilka, A. On the performance of reed-muller codes with respect to random errors and erasures. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1357–1376. SIAM, 2020.
- Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W., Bridgland, A., et al. Protein structure prediction using multiple deep neural networks in the 13th critical assessment of

- protein structure prediction (casp13). *Proteins: Structure, Function, and Bioinformatics*, 87(12):1141–1148, 2019.
- Shannon, C. E. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- Sybis, M., Wesolowski, K., Jayasinghe, K., Venkatasubramanian, V., and Vukadinovic, V. Channel coding for ultra-reliable low-latency communication in 5g systems. In *2016 IEEE 84th vehicular technology conference (VTC-Fall)*, pp. 1–5. IEEE, 2016.
- Tal, I. and Vardy, A. How to construct polar codes. *IEEE Trans. Inf. Theory*, 59(10):6562–6582, 2013.
- Tal, I. and Vardy, A. List decoding of polar codes. *IEEE Transactions on Information Theory*, 61(5):2213–2226, 2015.
- Tallini, L. and Cull, P. Neural nets for decoding error-correcting codes. In *IEEE Technical applications conference and workshops. Northcon/95. Conference record*, pp. 89. IEEE, 1995.
- Teng, C.-F., Wu, C.-H. D., Ho, A. K.-S., and Wu, A.-Y. A. 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)*, pp. 1413–1417. IEEE, 2019.
- Udrescu, S.-M. and Tegmark, M. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.
- Vasić, B., Xiao, X., and Lin, S. Learning to decode ldpc codes with finite-alphabet message passing. In *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9. IEEE, 2018.
- Wang, X., Zhang, H., Li, R., Huang, L., Dai, S., Huangfu, Y., and Wang, J. 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)*, pp. 1–5. IEEE, 2019.
- Welling, M. Neural augmentation in wireless communication, 2020.
- Xu, W., Wu, Z., Ueng, Y.-L., You, X., and Zhang, C. Improved polar decoder based on deep learning. In *2017 IEEE International workshop on signal processing systems (SiPS)*, pp. 1–6. IEEE, 2017.
- Ye, M. and Abbe, E. Recursive projection-aggregation decoding of reed-muller codes. *IEEE Transactions on Information Theory*, 66(8):4948–4965, 2020.

Appendix

A. Polar(64, 7) code

Recall from Section 5 that the Plotkin tree for a Polar code is obtained by freezing a set of leaves in a complete binary tree. These frozen leaves are chosen according to the reliabilities, or equivalently, error probabilities, of their corresponding bit channels. In other words, we first approximate the error probabilities of all the n -bit channels and pick the k -smallest of them using the procedure from (Tal & Vardy, 2013). These k active set of leaves correspond to the transmitted message bits, whereas the remaining $n - k$ frozen leaves always transmit zero.

Here we focus on a specific Polar code: Polar(64, 7), with code dimension $k = 7$ and blocklength $n = 64$. For Polar(64, 7), we obtain these active set of leaves to be $\mathcal{A} = \{48, 56, 60, 61, 62, 63, 64\}$, and the frozen set to be $\mathcal{A}^c = \{1, 2, \dots, 64\} \setminus \mathcal{A}$. Using these set of indices and simplifying the redundant branches, we obtain the Plotkin tree for Polar(64, 7) to be Figure 11. We observe that this Polar Plotkin tree shares some similarities with that of a RM(6, 1) code (with same $k = 7$ and $n = 64$) with key differences at the topmost and bottom most leaves.

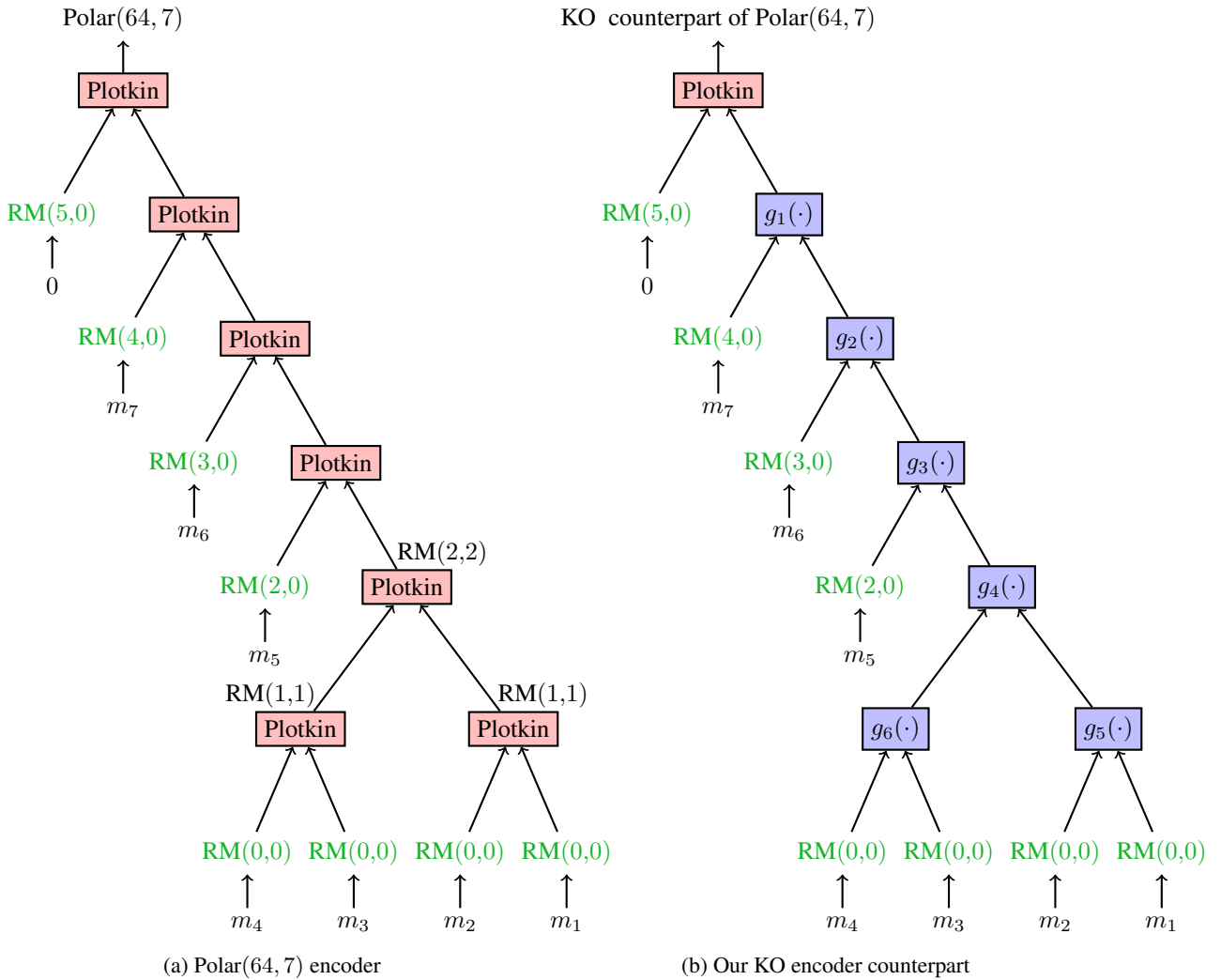


Figure 11. Plotkin trees for the Polar(64, 7) encoder and our neural KO encoder counterpart. Both codes have dimension $k = 7$ and blocklength $n = 64$.

Capitalizing on the encoding tree structure of Polar(64, 7), we build a corresponding KO encoder g_θ which inherits this tree skeleton. In other words, we generalize the Plotkin mapping blocks at the internal nodes of tree, except for the root node, and replace them with a corresponding neural network g_i . Figure 11 depicts the Plotkin tree of our KO encoder.

The KO decoder f_ϕ is designed similarly. Training of the (encoder, decoder) pair (g_θ, f_ϕ) is similar to that of the KO(8, 2) training which we detail in §4.

Figure 12 shows the BLER performance of the Polar(64, 7) code and its competing KO code, for the AWGN channel. Similar to the BER performance analyzed in Figure 10, the KO code is able to significantly improve the BLER performance. For example, we achieve a gain of around 0.5 dB when KO encoder is combined with the MAP decoding. Additionally, the close performance of the KO decoder to that of the MAP decoder confirms its optimality.

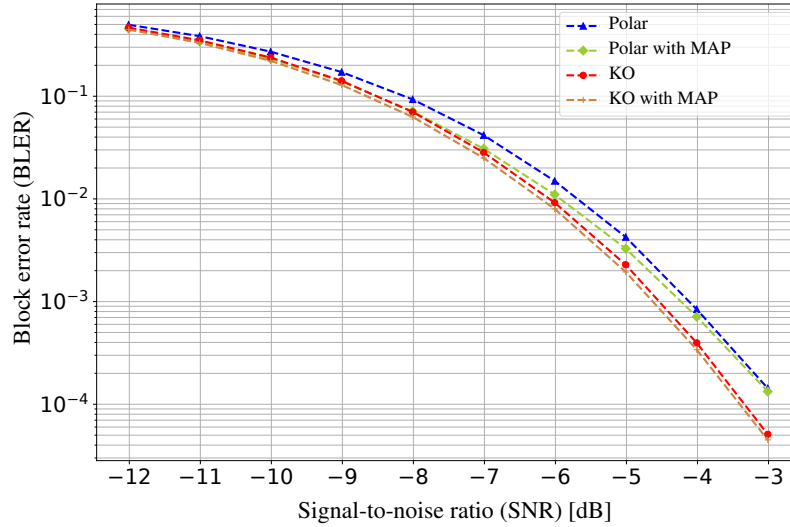


Figure 12. KO code achieves a significant gain over the Polar(64, 7) code in BLER when trained on AWGN channel. KO decoder also matches the optimal MAP decoder.

B. Gains with list decoding

Successive cancellation decoding can be significantly improved by list decoding. List decoding allows one to gracefully tradeoff computational complexity and reliability by maintaining a list (of a fixed size) of candidate codewords during the decoding process. The following figure demonstrates that KO(8,2) code with list decoding enjoys a significant gain over the non-list counterpart. This promising result opens several interesting directions, which are current focuses of active research.

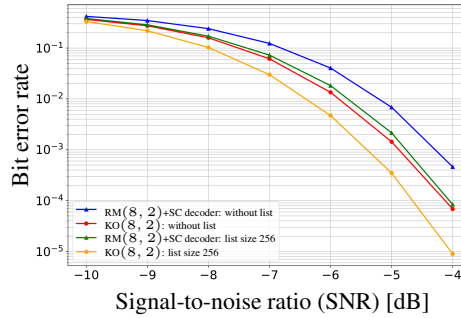


Figure 13. The same KO(8,2) encoder and decoder as those used in Figure 4 achieve a significant gain (without any retraining or fine-tuning) when list decoding is used together with the KO decoder. The magnitude of the gain is comparable to the gain achieved by the same list decoding technique on the successive cancellation decoder of the RM(8,2) code. We used the list decoding from (Dumer & Shabunov, 2006a) but without the permutation technique.

Polar codes with list decoding achieves the state-of-the-art performances (Tal & Vardy, 2015). It is a promising direction to design large block-lengths KO codes (based on the skeleton of Polar codes) that can improve upon the state-of-the-art list-decoded Polar codes. One direction is to train KO codes as we propose and include list decoding after the training. A more ambitious direction is to include list decoding in the training, potentially further improving the performance by discovering an encoder tailored for list decoding.

Unlike Polar codes, RM codes have an extra structure of an algebraic symmetry; a RM codebook is invariant under certain permutations. This can be exploited in list decoding as shown in (Dumer & Shabunov, 2006a), to get a further gain over what is shown in Figure 13. However, when a KO code is trained based on a RM skeleton, this symmetry is lost. A question of interest is whether one can discover nonlinear codes with such symmetry.

C. Discussion

C.1. On modulation and practicality of KO codes

We note that our real-valued KO codewords are *entirely practical* in wireless communication; the peak energy of a symbol is only 22.48% larger than the average in KO codes. The impact on the power amplifier is not any different from that of a more traditional modulation (like 16-QAM). Training KO code is a form of *jointly* designing the coding and modulation steps; this approach has a long history in wireless communication (e.g., Trellis coded modulation) but the performance gains have been restricted by the human ingenuity in constructing the heuristics.

C.2. Comparison with LDPC and BCH codes

We expect good performance for BCH at the short blocklengths considered in the paper though with high-complexity (polynomial time) decoders such as ordered statistics decoder (OSD). On the other hand, there does not exist good LDPC codes at the k and n regimes of this paper; thus, we do not expect good performance for LDPC at these regimes.

D. Plotkin construction

Plotkin (1960) proposed this scheme in order to combine two codes of smaller code lengths and construct a larger code with the following properties. It is relatively easy to construct a code with either a high rate but a small distance (such as sending the raw information bits directly) or a large distance but a low rate (such as repeating each bit multiple times). Plotkin construction combines such two codes of rates $\rho_u > \rho_v$ and distances $d_u < d_v$, to design a larger block length code satisfying rate $\rho = (\rho_u + \rho_v)/2$ and distance $\min\{2d_u, d_v\}$. This significantly improves upon a simple time-sharing of those codes, which achieves the same rate but distance only $\min\{d_u, d_v\}$.

Note: Following the standard convention, we fix the leaves in the Plotkin tree of a first order $\text{RM}(m, 1)$ code to be zeroth order RM codes and the full-rate $\text{RM}(1, 1)$ code. On the other hand, a second order $\text{RM}(m, 2)$ code contains the first order RM codes and the full-rate $\text{RM}(2, 2)$ as its leaves.

E. KO(8, 2): Architecture and training

As highlighted in §4, our KO codes improve upon RM codes significantly on a variety of benchmarks. We present the architectures of the KO(8, 2) encoder and the KO(8, 2) decoder, and their joint training methodology that are crucial for this superior performance.

E.1. KO(8, 2) encoder

Architecture. KO(8, 2) encoder inherits the same Plotkin tree structure as that of the second order $\text{RM}(8, 2)$ code and thus RM codes of first order and the second order $\text{RM}(2, 2)$ code constitute the leaves of this tree, as highlighted in Figure 14b. On the other hand, a critical distinguishing component of our KO(8, 2) encoder is a set of encoding neural networks $g_\theta = \{g_1, \dots, g_6\}$ that strictly generalize the Plotkin mapping. In other words, we associate a neural network $g_i \in g_\theta$ to each internal node i of this tree. If v and u denote the codewords arriving from left and right branches at this node, we combine them non-linearly via the operation $(u, v) \mapsto g_i(u, v)$.

We carefully parametrize each encoding neural network g_i so that they generalize the classical Plotkin map $\text{Plotkin}(u, v) =$

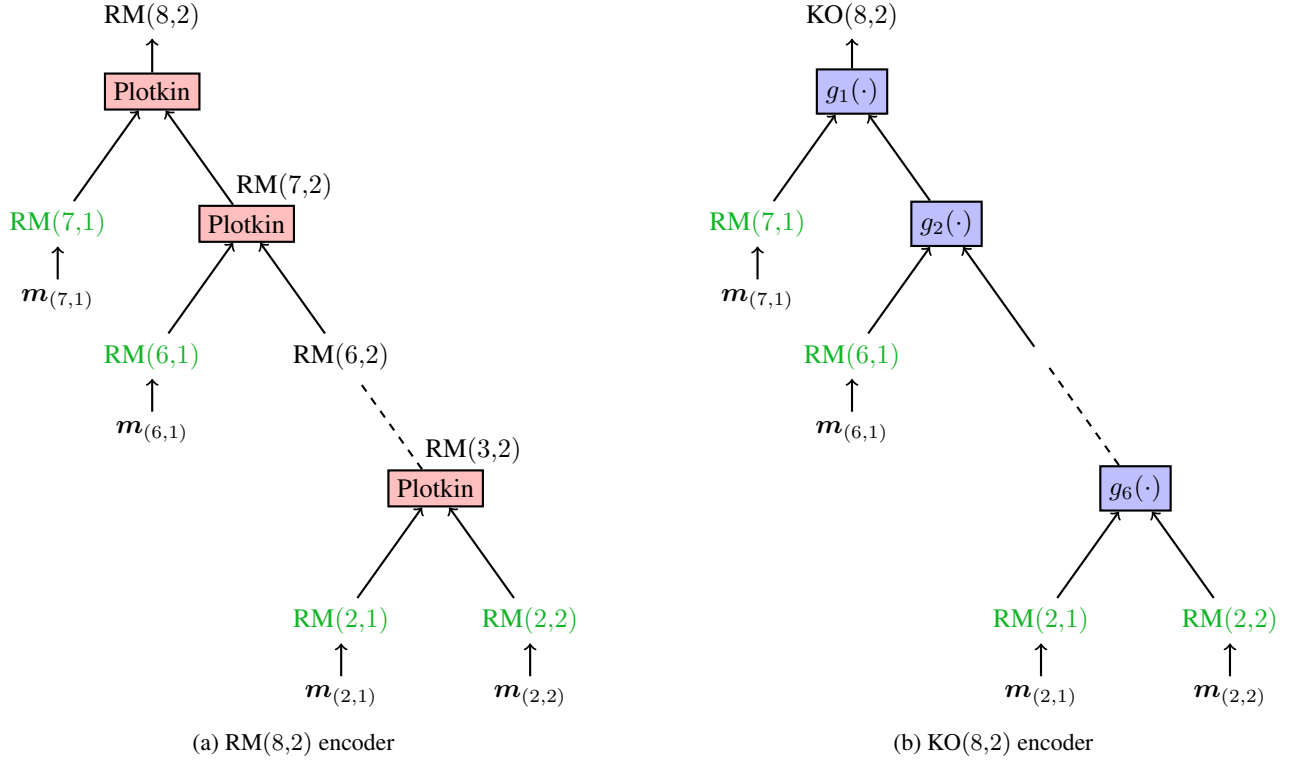


Figure 14. Plotkin trees for RM(8,2) and KO(8,2) encoders. Leaves are highlighted in green. Both codes have dimension $k = 37$ and blocklength $n = 256$.

$(\mathbf{u}, \mathbf{u} \oplus \mathbf{v})$. In particular, we represent them as $g_i(\mathbf{u}, \mathbf{v}) = (\mathbf{u}, \tilde{g}_i(\mathbf{u}, \mathbf{v}) + \mathbf{u} \oplus \mathbf{v})$, where $\tilde{g}_i : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a neural network of input dimension 2 and output size 1. Here \tilde{g}_i is applied coordinate-wise on its inputs \mathbf{u} and \mathbf{v} . This clever parametrization can also be viewed as a skip connection on top of the Plotkin map. Similar skip-like ideas have been successfully used in the literature though in a different context of learning decoders (Welling, 2020). On the other hand, we exploit these ideas for both encoders and decoders which further contribute to significant gains over RM codes.

Encoding. From an encoding perspective, recall that the KO(8, 2) code has code dimension $k = 37$ and block length $n = 256$. Suppose we wish to transmit a set of 37 message bits denoted as $\mathbf{m} = (\mathbf{m}_{(2,2)}, \mathbf{m}_{(2,1)}, \dots, \mathbf{m}_{(7,1)})$ through our KO(8, 2) encoder. We first encode the block of four message bits $\mathbf{m}_{(2,2)}$ into a RM(2, 2) codeword $\mathbf{c}_{(2,2)}$ using its corresponding encoder at the bottom most leaf of the Plotkin tree. Similarly we encode the next three message bits $\mathbf{m}_{(2,1)}$ into an RM(2, 1) codeword $\mathbf{c}_{(2,1)}$. We combine these codewords using the neural network g_6 at their parent node, which yields the codeword $\mathbf{c}_{(3,2)} = g_6(\mathbf{c}_{(2,2)}, \mathbf{c}_{(2,1)}) \in \mathbb{R}^8$. The codeword $\mathbf{c}_{(3,2)}$ is similarly combined with its corresponding left codeword and this procedure is thus recursively carried out till we reach the top most node of the tree, which outputs the codeword $\mathbf{c}_{(8,2)} \in \mathbb{R}^{256}$. Finally we obtain the unit-norm KO(8, 2) codeword \mathbf{x} by normalizing $\mathbf{c}_{(8,2)}$, i.e. $\mathbf{x} = \mathbf{c}_{(8,2)} / \|\mathbf{c}_{(8,2)}\|_2$.

Note that the map of encoding the message bits \mathbf{m} into the codeword \mathbf{x} , i.e. $\mathbf{x} = g_\theta(\mathbf{m})$, is differentiable with respect to θ since all the underlying operations at each node of the Plotkin tree are differentiable.

E.2. KO(8, 2) decoder

Architecture. Capitalizing on the recursive structure of the encoder, the KO(8, 2) decoder decodes the message bits from top to bottom, similar in style to Dumer’s decoding in §2. More specifically, at any internal node of the tree we first decode the message bits along its left branch, which we utilize to decode that of the right branch and this procedure is carried out recursively till all the bits are recovered. At the leaves, we use the Soft-MAP decoder to decode the bits.

Similar to the encoder g_θ , an important aspect of our KO(8, 2) decoder is a set of decoding neural networks $f_\phi = \{f_1, f_2, \dots, f_{11}, f_{12}\}$. For each node i in the tree, $f_{2i-1} : \mathbb{R}^2 \rightarrow \mathbb{R}$ corresponds to its left branch whereas $f_{2i} : \mathbb{R}^4 \rightarrow \mathbb{R}$

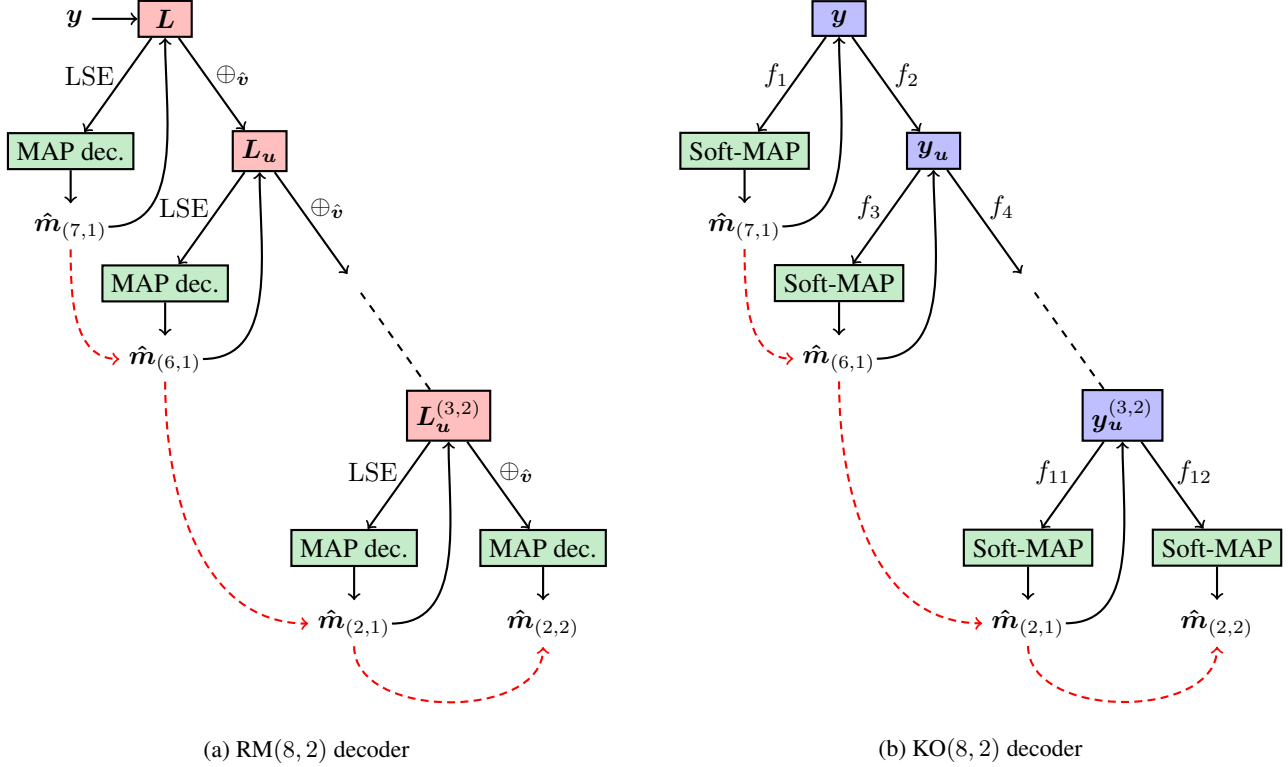


Figure 15. Plotkin trees for the RM(8, 2) and KO(8, 2) decoders. Red arrows indicate the bit decoding order.

corresponds to the right branch. The pair of decoding neural networks (f_{2i-1}, f_{2i}) can be viewed as matching decoders for the corresponding encoding network g_i : While g_i encodes the left and right codewords arriving at this node, the outputs of f_{2i-1} and f_{2i} represent appropriate Euclidean feature vectors for decoding them. Further, f_{2i-1} and f_{2i} can also be viewed as a generalization of Dumer's decoding to nonlinear real codewords: f_{2i-1} generalizes the LSE function, while f_{2i} extends the operation $\oplus_{\hat{v}}$. More precisely, we represent $f_{2i-1}(\mathbf{y}_1, \mathbf{y}_2) = \tilde{f}_{2i-1}(\mathbf{y}_1, \mathbf{y}_2) + \text{LSE}(\mathbf{y}_1, \mathbf{y}_2)$ whereas $f_{2i}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_v, \hat{v}) = \tilde{f}_{2i}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_v, \hat{v}) + \mathbf{y}_1 + (-1)^{\hat{v}} \mathbf{y}_2$, where $(\mathbf{y}_1, \mathbf{y}_2)$ are appropriate feature vectors from the parent node, and \mathbf{y}_v is the feature corresponding to the left-child v , and \hat{v} is the decoded left-child codeword. We explain about these feature vectors in more detail below. Note that both the functions \tilde{f}_{2i-1} and \tilde{f}_{2i} are also applied coordinate-wise.

Decoding. At the decoder suppose we receive a noisy codeword $\mathbf{y} \in \mathbb{R}^{256}$ at the root upon transmission of the actual codeword $\mathbf{x} \in \mathbb{R}^{256}$ along the channel. The first step is to obtain the LLR feature for the left RM(7, 1) codeword: we obtain this via the left neural network f_1 , i.e. $\mathbf{y}_v = f_1(\mathbf{y}_{1:128}, \mathbf{y}_{129:256}) \in \mathbb{R}^{128}$. Subsequently, the Soft-MAP decoder transforms this feature into an LLR vector for the message bits, i.e. $\mathbf{L}_{(7,1)} = \text{Soft-MAP}(f_1(\mathbf{y}_{1:128}, \mathbf{y}_{129:256}))$. Note that the message bits $\mathbf{m}_{(7,1)}$ can be hard decoded directly from the sign of $\mathbf{L}_{(7,1)}$. Instead here we use their soft version via the sigmoid function $\sigma(\cdot)$, i.e., $\hat{\mathbf{m}}_{(7,1)} = \sigma(\mathbf{L}_{(7,1)})$. Thus we obtain the corresponding RM(7, 1) codeword \hat{v} by encoding the message $\hat{\mathbf{m}}_{(7,1)}$ via an RM(7, 1) encoder. The next step is to obtain the feature vector for the right child. This is done using the right decoder f_2 , i.e. $\mathbf{y}_u = f_2(\mathbf{y}_{1:128}, \mathbf{y}_{129:256}, \mathbf{y}_v, \hat{v})$. Utilizing this right feature \mathbf{y}_u the decoding procedure is thus recursively carried out till we compute the LLRs for all the remaining message bits $\mathbf{m}_{(6,1)}, \dots, \mathbf{m}_{(2,2)}$ at the leaves. Finally we obtain the full LLR vector $\mathbf{L} = (\mathbf{L}_{(7,1)}, \dots, \mathbf{L}_{(2,2)})$ corresponding to the message bits \mathbf{m} . A simple sigmoid transformation, $\sigma(\mathbf{L})$, further yields the probability of each of these message bits being zero, i.e. $\sigma(\mathbf{L}) = \mathbb{P}[\mathbf{m} = \mathbf{0}]$.

Note that the decoding map $f_\phi : \mathbf{y} \mapsto \mathbf{L}$ is fully differentiable with respect to ϕ , which further ensures a differentiable loss for training the parameters (θ, ϕ) .

E.3. Training

Recall that we have the following flow diagram from encoder till the decoder when we transmit the message bits \mathbf{m} : $\mathbf{m} \xrightarrow{g_\theta} \mathbf{x} \xrightarrow{\text{Channel}} \mathbf{y} \xrightarrow{f_\phi} \mathbf{L} \xrightarrow{\sigma(\cdot)} \sigma(\mathbf{L})$. In view of this, we define an end-to-end differentiable cross entropy loss function to train the parameters (θ, ϕ) , i.e.

$$L(\theta, \phi) = \sum_j m_j \log(1 - \sigma(L_j)) + (1 - m_j) \log \sigma(L_j).$$

Finally we run Algorithm 1 on the loss $L(\theta, \phi)$ to train the parameters (θ, ϕ) via gradient descent.

F. Soft-MAP decoder

As discussed earlier (see also Figure 15), Dumer's decoder for second-order RM codes $\text{RM}(m, 2)$ performs MAP decoding at the leaves while our KO decoder applies Soft-MAP decoding at the leaves. The leaves of both $\text{RM}(m, 2)$ and $\text{KO}(m, 2)$ codes are comprised of order-one RM codes and the $\text{RM}(2, 2)$ code. In this section, we first briefly state the MAP decoding rule over general binary-input memoryless channels and describe how the MAP rule can be obtained in a more efficient way, with complexity $\mathcal{O}(n \log n)$, for first-order RM codes. We then present the generic Soft-MAP decoding rule and its efficient version for first-order RM codes.

MAP decoding. Given a length- n channel LLR vector $\mathbf{l} \in \mathbb{R}^n$ corresponding to the transmission of a given (n, k) code, i.e. code dimension is k and block length is n , with codebook \mathcal{C} over a general binary-input memoryless channel, the MAP decoder picks a codeword \mathbf{c}^* according to the following rule (Abbe et al., 2020)

$$\mathbf{c}^* = \underset{\mathbf{c} \in \mathcal{C}}{\text{argmax}} \quad \langle \mathbf{l}, 1 - 2\mathbf{c} \rangle, \quad (5)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner-product of two vectors. Obviously, the MAP decoder needs to search over all 2^k codewords while each time computing the inner-product of two length- n vectors. Therefore, the MAP decoder has a complexity of $\mathcal{O}(n2^k)$. Thus the MAP decoder can be easily applied to decode small codebooks like an $\text{RM}(2, 2)$ code, that has blocklength $n = 4$ and a dimension $k = 4$, with complexity $\mathcal{O}(1)$. On the other hand, a naive implementation of the MAP rule for $\text{RM}(m, 1)$ codes, that have $2^k = 2^{m+1} = 2n$ codewords, requires $\mathcal{O}(n^2)$ complexity. However, utilizing the special structure of order-1 RM codes, one can apply the fast Hadamard transform (FHT) to implement their MAP decoding in a more efficient way, i.e., with complexity $\mathcal{O}(n \log n)$. The idea behind the FHT implementation is that the standard $n \times n$ Hadamard matrix \mathbf{H} contains half of the the $2n$ codewords of an $\text{RM}(m, 1)$ code (in ± 1), and the other half are just $-\mathbf{H}$. Therefore, FHT of the vector \mathbf{l} , denoted by \mathbf{l}_{WH} , lists half of the $2n$ inner-products in (5), and the other half are obtained the as $-\mathbf{l}_{\text{WH}}$. Therefore, the FHT version of the MAP decoder for first-order RM codes can be obtained as

$$\mathbf{c}^* = (1 - \text{sign}(\mathbf{l}_{\text{WH}}(i^*))\mathbf{h}_{i^*})/2 \quad \text{s.t.} \quad i^* = \underset{i \in [n]}{\text{argmax}} \quad |\mathbf{l}_{\text{WH}}(i)|, \quad (6)$$

where $\mathbf{l}_{\text{WH}}(i)$ is the i -th element of the vector \mathbf{l}_{WH} , and \mathbf{h}_i is the i -th row of the matrix \mathbf{H} . Given that \mathbf{l}_{WH} can be efficiently computed with $\mathcal{O}(n \log n)$ complexity, the FHT version of the MAP decoder for the first-order RM codes, described in (6), has a complexity of $\mathcal{O}(n \log n)$.

Soft-MAP. Note that the MAP decoder and its FHT version involve $\text{argmax}(\cdot)$ operation which is not differentiable. In order to overcome this issue, we obtain the soft-decision version of the MAP decoder, referred to as Soft-MAP decoder, to come up with differentiable decoding at the leaves (Jamali et al., 2021). The Soft-MAP decoder obtains the soft LLRs instead of hard decoding of the codes at the leaves. Particularly, consider an AWGN channel model as $\mathbf{y} = \mathbf{s} + \mathbf{n}$, where \mathbf{y} is the length- n vector of the channel output, $\mathbf{s} := 1 - 2\mathbf{c}$, $\mathbf{c} \in \mathcal{C}$, and \mathbf{n} is the vector of the Gaussian noise with mean zero and variance σ^2 per element. The LLR of the i -th information bit u_i is then defined as

$$\mathbf{l}_{\text{inf}}(i) := \ln \left(\frac{\Pr(u_i = 0 | \mathbf{y})}{\Pr(u_i = 1 | \mathbf{y})} \right). \quad (7)$$

By applying the Bayes' rule, the assumption of $\Pr(u_i = 0) = \Pr(u_i = 1)$, the law of total probability, and the distribution of the Gaussian noise, we can write (7) as

$$\mathbf{l}_{\text{inf}}(i) = \ln \left(\frac{\sum_{\mathbf{s} \in \mathcal{C}_i^0} \exp(-\|\mathbf{y} - \mathbf{s}\|_2^2 / \sigma^2)}{\sum_{\mathbf{s} \in \mathcal{C}_i^1} \exp(-\|\mathbf{y} - \mathbf{s}\|_2^2 / \sigma^2)} \right). \quad (8)$$

We can also apply the max-log approximation to approximate (8) as follows.

$$l_{\text{inf}}(i) \approx \frac{1}{\sigma^2} \min_{c \in \mathcal{C}_i^1} \|\mathbf{y} - \mathbf{s}\|_2^2 - \frac{1}{\sigma^2} \min_{c \in \mathcal{C}_i^0} \|\mathbf{y} - \mathbf{s}\|_2^2, \quad (9)$$

where \mathcal{C}_i^0 and \mathcal{C}_i^1 denote the subsets of codewords that have the i -th information bit u_i equal to zero and one, respectively. Finally, given that the length- n LLR vector of the channel output can be obtained as $\mathbf{l} := 2\mathbf{y}/\sigma^2$ for the AWGN channels, and assuming that all the codewords \mathbf{s} 's have the same norm, we obtain a more useful version of the Soft-MAP rule for approximating the LLRs of the information bits as

$$l_{\text{inf}}(i) \approx \max_{c \in \mathcal{C}_i^0} \langle \mathbf{l}, 1 - 2c \rangle - \max_{c \in \mathcal{C}_i^1} \langle \mathbf{l}, 1 - 2c \rangle. \quad (10)$$

It is worth mentioning at the end that, similar to the MAP rule, one can compute all the 2^k inner products in $\mathcal{O}(n2^k)$ time complexity, and then obtain the soft LLRs by looking at appropriate indices. As a result, the complexity of the Soft-MAP decoding for decoding RM($m, 1$) and RM($2, 2$) codes is $\mathcal{O}(n^2)$ and $\mathcal{O}(1)$ respectively. However, one can apply an approach similar to (6) to obtain a more efficient version of the Soft-MAP decoder, with complexity $\mathcal{O}(n \log n)$, for decoding RM($m, 1$) codes.

G. Experimental details

We provide our code at <https://github.com/deepcomm/KOcodes>.

G.1. Training algorithm

Algorithm 1: Training algorithm for KO(8,2)

Input: number of epochs T , number of encoder training steps T_{enc} , number of decoder training steps T_{dec} , encoder training SNR SNR_{enc} , decoder training SNR SNR_{dec} , learning rate for encoder lr_{enc} , learning rate for decoder lr_{dec}

```

1 Initialize  $(\theta, \phi)$ 
2 for  $T$  steps do
3   for  $T_{\text{dec}}$  steps do
4     Generate a minibatch of random message bits  $\mathbf{m}$ 
5     Simulate AWGN channel with  $\text{SNR}_{\text{dec}}$ 
6     Fix  $\theta$ , update  $\phi$  by minimizing  $L(\theta, \phi)$  using Adam with learning rate  $\text{lr}_{\text{dec}}$ 
7   for  $T_{\text{enc}}$  steps do
8     Generate a minibatch of random message bits  $\mathbf{m}$ 
9     Simulate AWGN channel with  $\text{SNR}_{\text{enc}}$ 
10    Fix  $\phi$ , update  $\theta$  by minimizing  $L(\theta, \phi)$  using Adam with learning rate  $\text{lr}_{\text{enc}}$ 
```

Output: (θ, ϕ)

G.2. Hyper-parameter choices for KO(8,2)

We choose batch size $B = 50000$, encoder training SNR $\text{SNR}_{\text{enc}} = -3\text{dB}$, decoder training SNR $\text{SNR}_{\text{dec}} = -5\text{dB}$, number of epochs $T = 2000$, number of encoder training steps $T_{\text{enc}} = 50$, number of decoder training steps $T_{\text{dec}} = 500$. For Adam optimizer, we choose learning rate for encoder $\text{lr}_{\text{enc}} = 10^{-5}$ and for decoder $\text{lr}_{\text{dec}} = 10^{-4}$.

G.3. Neural network architecture of KO(8,2)

G.3.1. INITIALIZATION

We design our (encoder, decoder) neural networks to generalize and build upon the classical (Plotkin map, Dumer's decoder). In particular, as discussed in Section E.1, we parameterize the KO encoder g_θ , as $g_i(\mathbf{u}, \mathbf{v}) = (\mathbf{u}, \tilde{g}_i(\mathbf{u}, \mathbf{v}) + \mathbf{u} \oplus \mathbf{v})$, where $\tilde{g} : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a fully connected neural network, which we delineate in Section G.3.2. Similarly, for KO decoder, we parametrize

it as $f_{2i-1}(\mathbf{y}_1, \mathbf{y}_2) = \tilde{f}_{2i-1}(\mathbf{y}_1, \mathbf{y}_2) + \text{LSE}(\mathbf{y}_1, \mathbf{y}_2)$ and $f_{2i}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_v, \hat{\mathbf{v}}) = \tilde{f}_{2i}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_v, \hat{\mathbf{v}}) + \mathbf{y}_1 + (-1)^{\hat{\mathbf{v}}} \mathbf{y}_2$, where $\tilde{f}_{2i-1} : \mathbb{R}^2 \rightarrow \mathbb{R}$ and $\tilde{f}_{2i} : \mathbb{R}^4 \rightarrow \mathbb{R}$ are also fully connected neural networks whose architectures are described in Section G.3.4 and Section G.3.3. If $\tilde{f} \approx 0$ and $\tilde{g} \approx 0$, we are able to thus recover the standard RM(8, 2) encoder and its corresponding Dumer decoder. By initializing all the weight parameters (θ, ϕ) sampling from $\mathcal{N}(0, 0.02^2)$, we are able to approximately recover the performance RM(8, 2) at the beginning of the training which acts as a good initialization for our algorithm.

G.3.2. ARCHITECTURE OF \tilde{g}_i

- Dense(units=2 × 32)
- SeLU()
- Dense(units=32 × 32)
- SeLU()
- Dense(units=32 × 32)
- SeLU()
- Dense(units=32 × 1)

G.3.3. ARCHITECTURE OF \tilde{f}_{2i}

- Dense(units=4 × 32)
- SeLU()
- Dense(units=32 × 32)
- SeLU()
- Dense(units=32 × 32)
- SeLU()
- Dense(units=32 × 1)

G.3.4. ARCHITECTURE OF \tilde{f}_{2i-1}

- Dense(units=2 × 32)
- SeLU()
- Dense(units=32 × 32)
- SeLU()
- Dense(units=32 × 32)
- SeLU()
- Dense(units=32 × 1)

H. Results for Order-1 codes

Here we focus on first order KO($m, 1$) codes, and in particular KO(6, 1) code that has code dimension $k = 7$ and blocklength $n = 64$. The training of the (encoder, decoder) pair (g_θ, f_ϕ) for KO(6, 1) is almost identical to that of the second order RM(8, 2) described in §3. The only difference is that we now use the Plotkin tree structure of the corresponding RM(6, 1) code. In addition, we also train our neural encoder g_θ together with the differentiable MAP decoder, i.e. the Soft-MAP, to compare its performance to that of the RM codes. Figure 16 illustrates these results.

The left panel of Figure 16 highlights that KO(6, 1) obtains significant gain over RM(6, 1) code (with Dumer decoder) when both the neural encoder and decoder are trained jointly. On the other hand, in the right panel, we notice that we match the performance of that of the RM(6, 1) code (with the MAP decoder) when we just train the encoder g_θ (with the MAP decoder). In other words, under the optimal MAP decoding, KO(6, 1) and RM(6, 1) codes behave the same. Note that

the only caveat for $\text{KO}(6, 1)$ in the second setting is that its MAP decoding complexity is $O(n^2)$ while that of the RM is $O(n \log n)$.

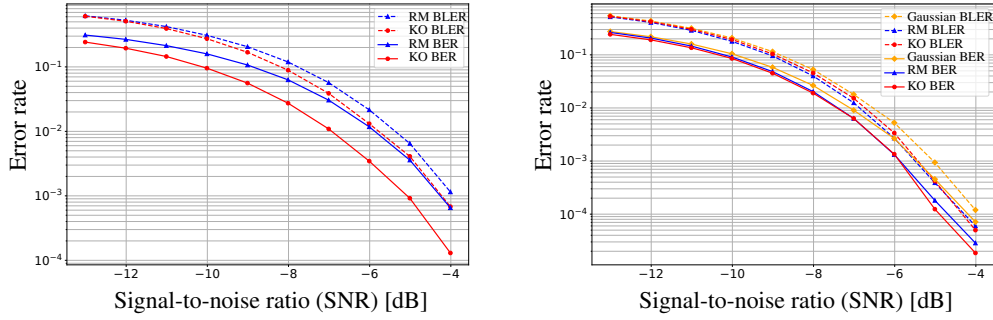


Figure 16. $\text{KO}(6, 1)$ code. **Left:** $\text{KO}(6, 1)$ code achieves significant gain over $\text{RM}(6, 1)$ code (with Dumer) when trained on AWGN channel. **Right:** Under the optimal MAP decoding, $\text{KO}(6, 1)$ and $\text{RM}(6, 1)$ codes achieve the same performance. Error rates for a random Gaussian codebook are also plotted as a baseline.