Interpreting KO Codes

R. Shekhar, N. Devroye, Gy. Turán*, and M. Žefran
University of Illinois Chicago, Chicago, IL, USA
*HUN-REN-SZTE Research Group on AI
{rshekh3, gyt, mzefran, devroye}@uic.edu

***Abstract*—The KO (Kronecker Operation) code is a recent deep-learned error-correcting code using a neural network architecture to generalize a Reed-Muller code with Dumer decoding. Analyzing the encoder modules and using ablation techniques, we give interpretations of the KO encoder which significantly reduce the number of parameters. We also discuss interpretability aspects of the KO decoder. The interpretation opens up possibilities to give an explicit representation of KO codes, which could be useful for more efficient learning of KO codes and explaining the learning mechanism underlying the empirical observations made about its performance in previous work.**

I. Introduction

Deep learning has been used recently to obtain error-correcting codes, both for point-to-point AWGN channels and for channels with feedback. Interpreting these codes is discussed in several papers as a natural question raised by the black-box nature of these codes [1], [2], which is also relevant for the potential practical applicability. Interpretations could be useful for generalizing the learned networks to other parameters and to understand their relationship to known codes. A systematic study of interpretability of deep-learned error-correcting codes is initiated in [3]–[6].

The architecture of neural networks learning error-correcting codes is often designed to reflect the encoder and decoder structure of some known family of codes. For example, [7]'s architecture mimics that of Turbo codes, and [2]'s architecture that of Reed-Muller (RM) codes. This technique of "neuralizing" the family can be viewed as trying to find an optimal code in the space of codes "resembling" codes in the family. The possibility of optimization by deep learning thus provides new opportunities not only for constructing new codes, but also for exploring properties of the spaces of existing codes such as local optimality of a certain code.

The Reed-Muller (RM) code, discovered 70 years ago, is among the most classical. A survey, including recent results on achieving capacity on certain channels, is given by [8]. The deep-learned KO code of [2] is a neuralization of the RM code, and it improves upon RM codes with Dumer decoding both in terms of BER and BLER for second-order RM codes of lengths 256 and 512. The number of parameters of the KO network was reduced by about half and its training was accelerated in [9].

One explanation for the improvements given by KO codes given by [2] is that being real-valued, they combine encoding and modulation (this holds for most deep-learned codes for AWGN channels in general). They also show that the distribution of pairwise distances (corresponding to the weight distribution for linear codes) is quite similar to the Gaussian codebook. Another interesting observation concerns the unequal contribution of various branches to the BLER of the Plotkin tree for RM. It is shown in [2] that KO achieves an improvement of the balance.

In this paper, we further explore the interpretation of KO codes. We focus on interpreting the encoder. We take a closer look "under the hood" by analyzing the modules g_i of the $KO(8,2)$ encoder. The main result is that the encoder can be simplified significantly. This result is obtained using two different interpretation techniques.

The first approach is to analyze the modules, by simulating the KO network on a random sample and tracking the activations produced. The technique as described applies to second-order RM only, but it could possibly be extended to higher orders as well. It turns out that the activations exhibit a sharp clustering. This allows the replacement of the modules by small lookup tables. The performance of the simplified encoder paired with the learned decoder without any change is very similar to the original network.

The second approach is an ablation technique replacing modules in the lower layers with the original RM modules, and keeping only the two top ones. The inputs to the remaining modules are binary, and therefore these modules can also be replaced by small lookup tables. Pairing this simplified encoder with the learned decoder without any change gives again a very similar performance to the original network. Since the inputs to functions g_i are discrete, we can use a finite number of constants to specify these functions. When the top two functions are used, this would result in 12 constants. However, as observed above, the support of the functions collapses, and instead of 12, we just need 8 constants. Ultimately, this network reduces the number of parameters in the encoder from more than 13,000 to 8.

We also consider stitching or "Frankensteining" different encoders and decoders [10], [11]. This technique was used in previous work on interpreting deep-learned error-correcting

Non-student authors are in alphabetic order. This work was supported by NSF under awards 2217023, 2240532 and 1828265, and by the AI National Laboratory Program (RRF-2.3.1-21-2022-00004). The contents of this article are solely the responsibility of the authors and do not necessarily represent the official views of the NSF.

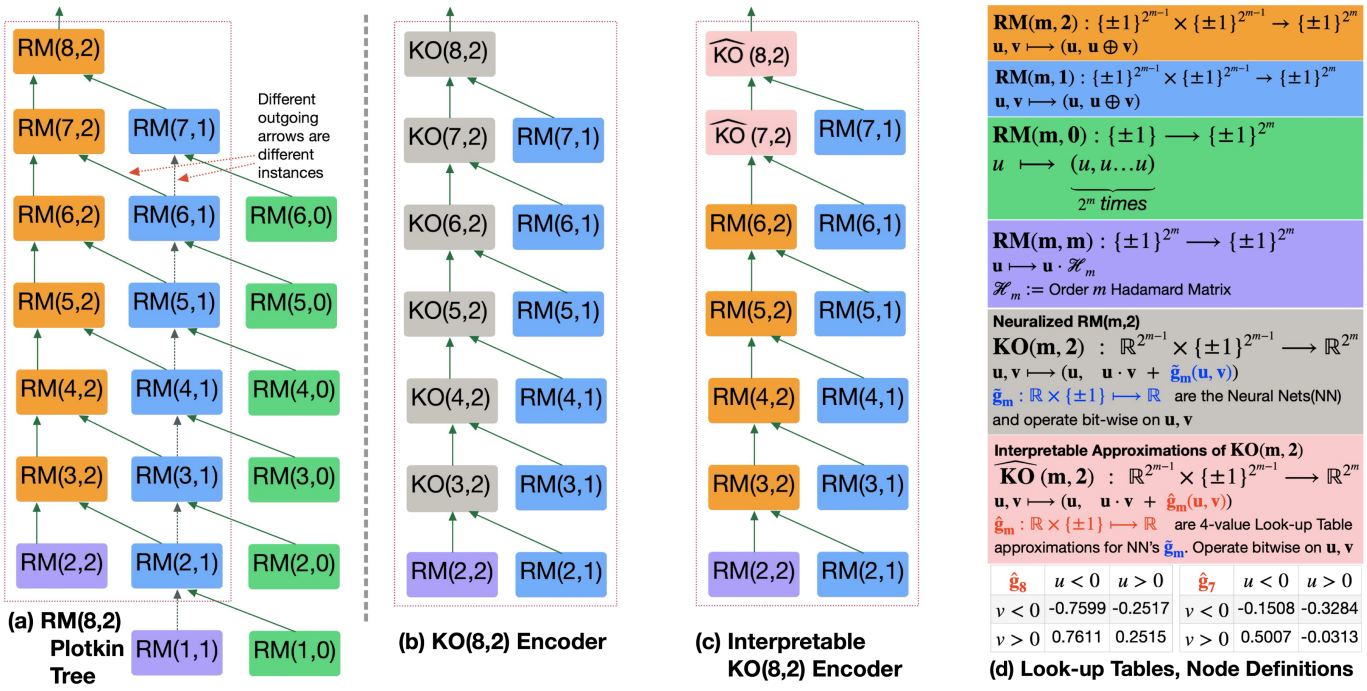

Fig. 1: **(a)** RM(8,2) Encoder **(b)** KO(8,2) construction using RM(8,2). The orange *(order-2)* and blue *(order-1)* nodes are Plotkin nodes. The grey KO nodes add the trainable $\tilde{g}$ functions to the Plotkin nodes. *order-1* nodes(blue) are considered leaves, as they can be MAP decoded. **(c)** Interpretable KO(8,2) encoder. Replaces all the neuralized grey KO nodes in KO(8,2) with *interpretable* nodes - $\widehat{\text{KO}}$ (pink) or just plain *order-2* Reed Muller nodes (orange). **(d)** Look-up tables $\hat{g}_8, \hat{g}_7$ for the interpretable nodes $\widehat{\text{KO}}(8,2)$, $\widehat{\text{KO}}(7,2)$ respectively.

codes [3], [4] to draw conclusions about the relative importance of learned encoders and decoders. The encoders considered are RM, KO and the various interpretable encoders, and the decoders are the Dumer [12], RPA [13] and KO decoders. These experiments are used to evaluate the various interpreted encoders and to make some observations about decoder interpretabilty.

II. Reed-Muller (RM) Codes

Reed-Muller (RM) codes are a family of error-correcting codes defined by two parameters: m, the size of the code, and r, the order of the code. The parameters of the RM(m, r) code determine the length, dimension, rate, and minimum distance as follows: length $n = 2^m$, dimension $k = \sum_{i=0}^{r} \binom{m}{i}$, rate $\rho = k/n$, minimum distance $d = 2^{m-r}$.

A. RM Encoder

RM codes may be obtained using the recursive *Plotkin construction*, with basic building block $(\mathbf{u}, \mathbf{u} \oplus \mathbf{v})$, where $\mathbf{u}, \mathbf{v} \in \{0,1\}^{\ell}$, $\oplus$ represents bit-wise XOR, and $(\cdot, \cdot)$ denotes concatenation. This recursive process is used to construct RM(m, r) codes using the relation:

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

where RM$(m, 0)$ represents a repetition code that repeats a single information bit 2^m times, and RM(m, m) is a full-rate code with $k = n = 2^m$. The result of this recursive construction can be visualized as a *computation tree* as in Fig. 1(a), referred to as a *Plotkin tree*, where each non-leaf node corresponds to a Plotkin mapping, and the leaves represent repetition or full-rate codes at various recursion levels.

B. RM Decoder

RM codes support multiple decoding algorithms, with *Dumer's recursive decoding* [12], [14] being one of the most efficient. This decoding process takes advantage of the recursive structure defined by the Plotkin construction. At each node $RM(m, r)$ of the Plotkin tree, it obtains log-likelihood ratios (LLRs) to decode that stage. It uses the LLRs first to recursively decode the lower order branch $RM(m-1, r-1)$, and then uses the decoded codeword of that branch together with the LLRs to decode the higher order branch $RM(m-1, r)$.

III. KO Codes

KO codes introduce a new family of error-correcting codes by generalizing the recursive Plotkin construction used in RM codes. These codes leverage deep learning to discover nonlinear encoding and decoding mechanisms.

A. KO Encoder

The KO encoder builds upon the recursive Plotkin tree structure of RM codes. While RM codes rely on linear Plotkin mappings of the form $(u, v) \mapsto (u, u \oplus v)$, the KO encoder adds a neural network component to $u \oplus v$ enabling nonlinear transformations, see Fig. 1(b).

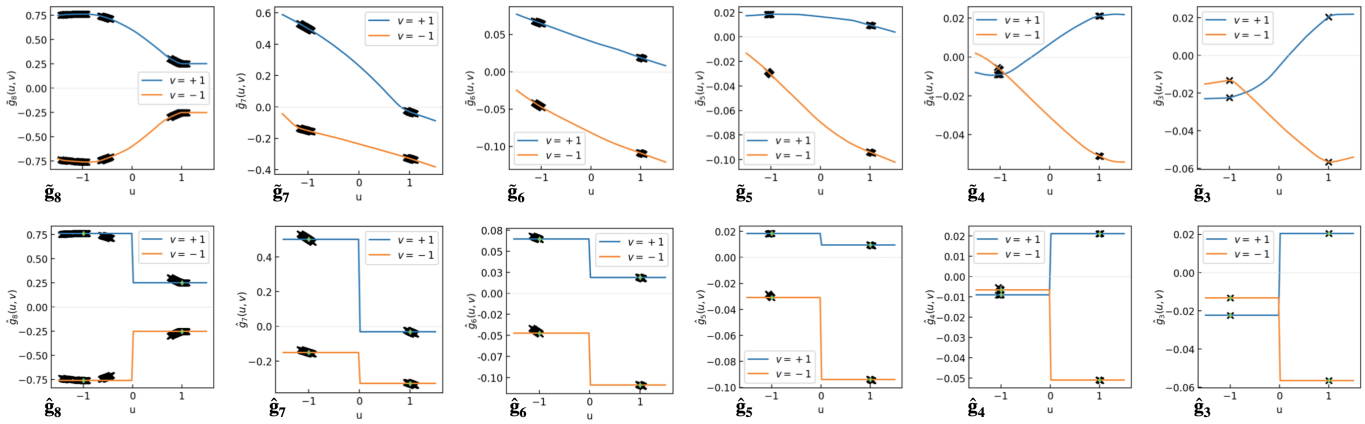

Fig. 2: **Row 1** : Plots of $\tilde{g}_i : \mathbb{R} \times \{\pm 1\} \longrightarrow \mathbb{R}$ implemented by the Neural Nets of KO Encoder. The black points represent the probability density of inputs seen by the $\tilde{g}$ functions during simulations with a large number of randomly generated inputs.
Row 2: Approximations $\hat{g}$ of $\tilde{g}$'s obtained by reducing the clusters to $(\pm 1, \pm 1)$, shown as green points, and replacing the neural nets(NN) with a lookup table with four values/cluster representatives. Each NN is replaced by four constants.

For each $RM(i,2)$ node in the Plotkin tree, KO encoder replaces it with $KO(i,2)$ node, in which the Plotkin operation $(\mathbf{u},\mathbf{v}) \longmapsto (\mathbf{u}, \mathbf{u} \oplus \mathbf{v})$ is replaced by its *neuralized* version $(\mathbf{u},\mathbf{v}) \longmapsto (\mathbf{u},\ \mathbf{u}\cdot\mathbf{v} + \tilde{g}_i(\mathbf{u},\mathbf{v}))$, where $\tilde{g}_i : \mathbb{R}\times\{\pm 1\} \longrightarrow \mathbb{R}$ is applied bitwise, and is implemented using a 4 layer fully connected neural net, with hidden layers of size 32 each, having 2241 learnable parameters. Since $u \in \mathbb{R}$, $u \oplus v$ in Plotkin nodes is replaced with $u \cdot v$ which are equivalent when $u, v \in \{\pm 1\}$. Since all these functions are applied bitwise, KO encoder maintains the same computational complexity as RM encoders.

Thus across all the stages the KO encoder has a total of 13,446 learnable parameters. This enables KO codes to explore a richer space of (possibly non-linear) encoders.

B. KO Decoder

The KO decoder's architecture is based on Dumer's recursive decoder for Reed Muller codes, but it enhances the Dumer decoder stages with neural nets to match the neural net encoders of the KO encoder. Specifically corresponding to each stage $KO(i+1,2)$ of encoder it adds two neural nets f_{i_1}, f_{i_2} in the Dumer's decoder to generalize the decoding of the $RM(i,1)$ and $KO(i,2)$ branch respectively.

The KO decoder retains the same $O(n \log n)$ decoding complexity as Dumer's decoder for RM codes.

It should be noted that our notation for the $\tilde{g}$,$\tilde{f}$ functions differs from the one used for the KO(8,2) code in [2], and mappings from one to the other can be read from table I.

C. Training

The neural net parameters θ, ϕ of the KO encoder g_θ and the KO decoder f_ϕ are trained end-to-end in a channel autoencoder setup to minimize the average bitwise binary cross entropy (BCE) which serves as a differentiable proxy for bit error rate (BER). Training is done using samples generated uniformly randomly for the inputs and from the appropriate normal distribution for the noise from the AWGN channel. The loss function is:

$$\mathcal{L}(\theta,\phi) = \mathbb{E}_{\substack{\mathbf{U}\sim\mathbb{F}_2^k \\ \mathbf{Z}\sim\mathcal{N}_{0,\sigma^2}^n}} \left[\frac{1}{k} \sum_{i=1}^{k} BCE(U_i, f_\phi(g_\theta(\mathbf{U}) + \mathbf{Z})_i) \right].$$

By introducing nonlinear transformations, KO codes expand the design space of encoding and decoding, enabling the possibility of *richer codebooks* like random Gaussian codebooks, known for their optimality, while also remaining computationally efficient to decode, unlike random codebooks.

$\tilde{g}_8$	$\tilde{g}_7$	$\tilde{g}_6$	$\tilde{g}_5$	$\tilde{g}_4$	$\tilde{g}_3$
$\tilde{g}_1$	$\tilde{g}_2$	$\tilde{g}_3$	$\tilde{g}_4$	$\tilde{g}_5$	$\tilde{g}_6$
$\tilde{f}_{7_1}, \tilde{f}_{7_2}$	$\tilde{f}_{6_1}, \tilde{f}_{6_2}$	$\tilde{f}_{5_1}, \tilde{f}_{5_2}$	$\tilde{f}_{4_1}, \tilde{f}_{4_2}$	$\tilde{f}_{3_1}, \tilde{f}_{3_2}$	$\tilde{f}_{2_1}, \tilde{f}_{2_2}$
$\tilde{f}_1, \tilde{f}_2$	$\tilde{f}_3, \tilde{f}_4$	$\tilde{f}_5, \tilde{f}_6$	$\tilde{f}_7, \tilde{f}_8$	$\tilde{f}_9, \tilde{f}_{10}$	$\tilde{f}_{11}, \tilde{f}_{12}$

TABLE I: Change of notation from KO(8,2) code in [2]. Rows 1,3 (grey) contain the names used in this paper; rows 2,4 contain the corresponding name used in [2].

IV. Encoder Interpretation

We proceed to open the black boxes of each of the neural networks $\tilde{g}$ in the KO encoder in Fig. 1, providing an explicit post-hoc interpretation of the learned encoding functions.

A. Visualizing the KO Encoder's Neural Nets

The modules $\tilde{g}_3, \tilde{g}_4, \ldots \tilde{g}_8$ in the KO encoder compute functions of the form $\mathbb{R} \times \{\pm 1\} \longrightarrow \mathbb{R}$, since the second input v is binary. Thus

$$\tilde{g}_i(u,v) = \mathbf{1}_{\{v=1\}} \cdot \tilde{g}_i(u,1) + \mathbf{1}_{\{v=-1\}} \cdot \tilde{g}_i(u,-1)$$

where $\mathbf{1}_{\{\cdot\}}$ denotes indicator function for condition $\{\cdot\}$.

Thus we can visualize $\tilde{g}_i$ as two *one variable* functions $\tilde{g}_i(u,+1)$ and $\tilde{g}_i(u,-1)$ by fixing the input v to $+1$ and -1 respectively. This allows an easy $2D$ visualization of the $\tilde{g}$ functions. All the $\tilde{g}_{8:3}(u,\pm 1)$ are plotted in Fig. 2, where each

$\tilde{g}_i$ is shown as two curves, blue and orange for $\tilde{g}_i(u, +1)$ and $\tilde{g}_i(u, -1)$, respectively.

B. Relevant Regions to Interpret

The visualizations are based on a random sample of inputs and the activations of the nodes for these inputs. We only need to interpret the $\tilde{g}$'s on the activated parts rather than the entire input domain, since only those points have been observed and are hence relevant. This will become important in the next subsection about approximation. Activated parts which have been seen seen during simulations / training - are plotted as black scatter points on the $\tilde{g}(u, \pm 1)$ curves in Fig. 2. It is remarkable how well the supports are clustered for all of the $\tilde{g}$'s, which makes them much easier to interpret, as we need to interpret them only around the black clusters on the curves.

C. Approximations

Due to clustering observed above, as the first approximation we replace the functions with a look up table containing same function value for each cluster. Thus the mapping looks like *input* $\rightarrow$ *cluster* $\rightarrow$ *$\tilde{g}$(cluster)*. With this we essentially end up with four input clusters for each $\tilde{g}_i(u, v)$ around $(\pm 1, \pm 1)$ and thus each $\tilde{g}_i$ is replaced by the four constants $\in \tilde{g}_i(\pm 1, \pm 1)$. We denote this resulting approximation as the corresponding $\hat{g}_i$, i.e.

$$\hat{g}_i(u, v) \quad := \quad (u, v) \longmapsto \tilde{g}_i(\text{sign}(u), \text{sign}(v))$$

We keep the decoder neural nets unmodified.

Even with this perhaps simplistic approximation of the KO encoder, we get a code which is almost identical in performance to the trained KO(8,2), see "KO(8,2) approx" in Fig. 4. Thus, perhaps surprisingly, each neural net module $\tilde{g}_i$ of more than 2,000 parameters can be interpreted as $\hat{g}_i$, which in turn is just a lookup table of four constants.

D. Interpretations from Ablations

We further try to evaluate how much of the performance of KO codes can be attributed to each stage from $3 \ldots 8$. To do this we define ablations of KO(8,2) as following:

$$\begin{aligned} \textbf{RM(8,2) + } \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \ldots := \text{RM(8,2) Encoder} \quad & + \text{Dumer Decoder} \\ + \tilde{g}_a \quad & + \tilde{f}_{(a-1)_1}, \tilde{f}_{(a-1)_2} \\ + \tilde{g}_b \quad & + \tilde{f}_{(b-1)_1}, \tilde{f}_{(b-1)_2} \\ \vdots & \end{aligned}$$

So under this definition KO(8,2) = **RM(8,2) + $\tilde{\mathbf{3}}, \tilde{\mathbf{4}} \ldots \tilde{\mathbf{8}}$**. We evaluate the ablations in the sequence RM(8,2); **RM(8,2) + $\tilde{\mathbf{8}}$**; **RM(8,2) + $\tilde{\mathbf{8}}, \tilde{\mathbf{7}}$**; $\cdots$ KO(8,2). We found that on this trajectory from RM(8,2) to KO(8,2), there are only two significant jumps in the code quality, first with **RM(8,2) + $\tilde{\mathbf{8}}$** and another with **RM(8,2) + $\tilde{\mathbf{8}}, \tilde{\mathbf{7}}$**, which matches the performance of KO(8,2).

This suggests that the rest of the stages $\tilde{\mathbf{6}} : \tilde{\mathbf{3}}$ have no significant impact on the performance of the KO(8,2) code. This is further validated by the magnitude of the outputs of $\tilde{g}$ functions on the black clusters in Fig. 2. For $\tilde{g}_8, \tilde{g}_7$, the outputs are comparable to $u \oplus v \in \{\pm 1\}$ to which they are added.

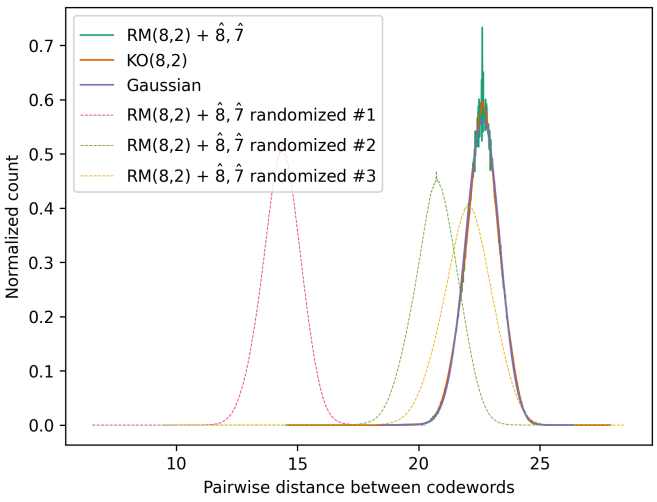

Fig. 3: Comparison of Distribution of Pairwise Distances of the codewords with that for a Random Gaussian Codebook. Similar to KO(8,2), **RM(8,2) + $\hat{\mathbf{8}}, \hat{\mathbf{7}}$** gets close to a random gaussian codebook like distribution, Even randomized **RM(8,2) + $\hat{\mathbf{8}}, \hat{\mathbf{7}}$** get similar shape but smaller mean distance.

However for $\tilde{g}_{3:6}$ the outputs are almost insignificant compared to ± 1, rendering them of little importance. Thus for all intents and purposes KO(8,2) is same as **RM(8,2) + $\tilde{\mathbf{8}}, \tilde{\mathbf{7}}$**.

This together with the approximation results from the previous subsection suggest that a further, and far more interpretable encoder should exist in the form of **RM(8,2) + $\hat{\mathbf{8}}, \hat{\mathbf{7}}$**, obtained by replacing $\tilde{g}_8, \tilde{g}_7$ in **RM(8,2) + $\tilde{\mathbf{8}}, \tilde{\mathbf{7}}$** with their interpretable versions $\hat{g}_8, \hat{g}_7$. **RM(8,2) + $\hat{\mathbf{8}}$** is exactly the same as **RM(8,2) + $\tilde{\mathbf{8}}$**, because in this configuration, $\tilde{g}_8$ only sees $(\pm 1, \pm 1)$ as inputs coming from the preceding stages. So it always produces the same output as $\hat{g}_8$.

We evaluated the performance of all these interpretable versions as well, which can be seen in Fig. 4. Here we indeed find that **RM(8,2) + $\hat{\mathbf{8}}, \hat{\mathbf{7}}$** *(green)* is responsible for the performance of KO codes, with **RM(8,2) + $\hat{\mathbf{8}}$** *(magenta)* as a stepping stone on the trajectory from **RM(8,2)** *(blue)* to **KO(8,2)** *(grey)*. Thus we found very simple and completely interpretable encoder for the KO(8,2) encoder. Fig. 1(c) shows the interpretable encoder in **RM(8,2) + $\hat{\mathbf{8}}, \hat{\mathbf{7}}$**.

E. Pairwise Distance Distribution of Codewords

It is noted in [2] as an indication of the improved quality of the KO code that the pairwise distance distribution of its codewords closely matches the distribution for a Gaussian codebook. The approximate encoder $\mathbf{RM}(\mathbf{8}, \mathbf{2}) + \hat{\mathbf{8}}, \hat{\mathbf{7}}$ also produces a similar distance distribution, see Fig. 3. To determine whether the precise values of the constants in $\hat{g}_8, \hat{g}_7$ mattered, we also randomly picked these constants from a standard normal distribution. Under random choices, the pairwise distance distribution still has a similar shape, but the distribution peaks before that of the KO encoder and the approximations. As lower mean distances are associated with worse codes, this again indicates that optimizing the BCE

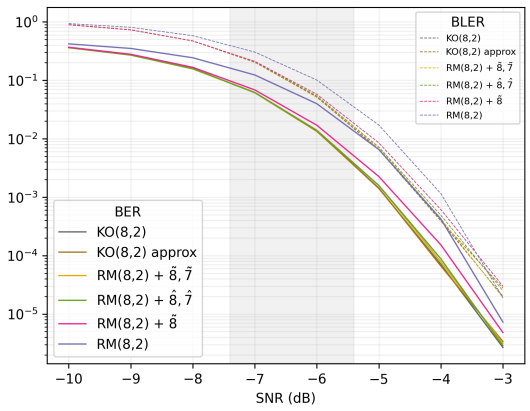

Fig. 4: BER plots for KO codes and its approximations.

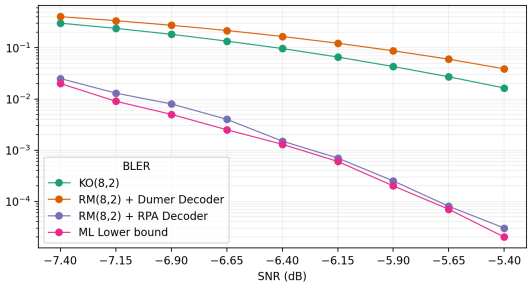

Fig. 5: Comparison of RM(8,2)+Dumer and KO(8,2) with (a) RM(8,2) + RPA Decoding proposed in [13], and (b) Lower bound on ML Decoding for RM(8,2). Plots for (a,b) taken from [13]. The SNR range corresponds to the shaded region in Fig. 4.

appears to also yield larger mean pairwise distances. One could then imagine training learned codes using some function of their distance distribution - for e.g. maximizing *mean squared pairwise distance* between the codewords

$$\mathbb{E}_{\substack{U\sim\mathbb{F}_2^k \\ V\sim\mathbb{F}_2^k}} \|g_\theta(U) - g_\theta(V)\|_2^2$$

- as an alternative encoder training objective (decoders could still be trained using BCE if an alternating encoder/decoder training is used). However, initial experiments with this setup for a smaller KO(4,2) code with ML Decoder did not yield any codes better than RM(4,2), hence it is not enough to just maximize the pairwise distances to get a good code, at least for small block lengths.

V. Decoder Insights

Decoder interpretation is typically more difficult than encoder interpretation, especially since closed-form expressions for ML decoders can only be computed in rare cases, and are certainly not known for RM codes. We thus limit our interpretation of the KO decoder to some observations, a number of which simply restate what is known from the literature.

We start with the original justification of KO codes from [2]: the KO scheme (KO Encoder + KO Decoder) outperforms the RM+Dumer scheme. Our experiments, shown in Fig. 5, confirm this. Further, since the KO decoder uses a recursive algorithm that mimics the Dumer decoder, the KO scheme is computationally efficient.

On the other hand, KO scheme does not match the performance of RM+ML (RM encoder + ML decoder) scheme. Again, this is clearly shown in Fig. 5. While an efficient ML decoder for RM is not known, it has been shown in [13] that the RPA with list decoder approaches the ML performance for RM(8,2). Fig. 5 shows the RPA decoder without list decoding, which also approximates ML decoding in this setting.

While there are hypotheses about why the KO scheme outperforms the RM+Dumer scheme (see Sec. IV-E), one wonders whether the KO decoder, when paired with the RM encoder (with the functions f_i appropriately retrained), could provide some benefit. It is not difficult to see that this is actually not the case as the Dumer decoder is optimal once Dumer's recursive decoding scheme is adopted. Our experiments, where we tried to train the KO decoder for the RM encoder, confirm this.

Finally, one wonders whether the KO decoder can be explicitly constructed (and interpreted) similarly to the Dumer decoder once the functions g_i are known (as in our interpretation). This turns out to be problematic due to the power normalization employed by the KO encoder. The RM codewords lie on a sphere by default because they are binary. The KO encoder makes the codewords real, and pushes them off a sphere so normalization is used to project them back. But this normalization is codeword dependent and is thus difficult to account for when trying to construct the KO decoder explicitly.

VI. Conclusions and Future Work

We continued the interpretation of KO codes [2] by significantly simplifying the encoder and discussing possibilities to interpret the decoder. As a result, we obtained an interpretable modification of KO(8,2) encoder using just 8 constants.

An interesting question is whether the interpretations can be used for more efficient learning of (second or higher-order) KO codes. Another question raised by the interpretable code is whether it can be represented in an analytically tractable form, which then perhaps could be used to explain experimentally observed properties of the KO code such as its error-balancing property mentioned in the introduction and the local (non)-optimality of RM(8,2). The interpretation also suggests directions for further experiments, such as analyzing influences and developing alternative neuralizations of Reed-Muller codes. The analysis presented in this paper is limited to the KO(8,2) code, and it remains to be seen how much these methods and especially interpretations will carry over to other KO code sizes, i.e. will these conclusions hold for KO(9,2)?

Acknowledgement We would like to thank Andreea Szőcs for useful discussions.

REFERENCES

[1] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," *NeurIPS*, vol. 31, 2018.

[2] 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.

[3] N. Devroye, N. Mohammadi, A. Mulgund, H. Naik, R. Shekhar, G. Turán, Y. Wei, and M. Žefran, "Interpreting deep-learned error-correcting codes," in *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2022, pp. 2457–2462.

[4] N. Devroye, A. Mulgund, R. Shekhar, G. Turán, M. Žefran, and Y. Zhou, "Interpreting training aspects of deep-learned error-correcting codes," in *2023 IEEE International Symposium on Information Theory (ISIT)*, 2023, pp. 2374–2379.

[5] Y. Zhou, N. Devroye, G. Turán, and M. Žefran, "Interpreting deepcode, a learned feedback code," in *2024 IEEE International Symposium on Information Theory (ISIT)*, 2024, pp. 1403–1408.

[6] Y. Zhou, N. Devroye, G. Turan, and M. Zefran, "Higher-order interpretations of deepcode, a learned feedback code," in *2024 60th Annual Allerton Conference on Communication, Control, and Computing*, 2024, pp. 1–8.

[7] 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 *NIPS*, Dec. 2019, pp. 2758–2768.

[8] E. Abbe, O. Sberlo, A. Shpilka, M. Ye *et al.*, "Reed-Muller codes," *Foundations and Trends® in Communications and Information Theory*, vol. 20, no. 1–2, pp. 1–156, 2023.

[9] S. Srivastava and A. Banerjee, "Dense KO codes: Faster convergence and reduced complexity through dense connectivity," in *IEEE International Symposium on Information Theory, ISIT 2024*. IEEE, 2024, pp. 1415–1420.

[10] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," *Int. J. Comput. Vis.*, vol. 127, no. 5, pp. 456–476, 2019.

[11] A. Csiszárik, P. Korösi-Szabó, Á. K. Matszangosz, G. Papp, and D. Varga, "Similarity and matching of neural network representations," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, 2021, pp. 5656–5668.

[12] I. Dumer and K. Shabunov, "Recursive decoding of Reed-Muller codes," in *2000 IEEE International Symposium on Information Theory (Cat. No.00CH37060)*, 2000, pp. 63–.

[13] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 4948–4965, 2020.

[14] I. Dumer, "Soft-decision decoding of Reed-Muller codes: a simplified algorithm," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, 2006.