Interpreting Training Aspects of Deep-Learned Error-Correcting Codes

Natasha Devroye, Abhijeet Mulgund, Raj Shekhar, György Turán*, Milos Žefran, and Yingyao Zhou
University of Illinois Chicago (UIC), Chicago, IL, USA
* UIC and MTA-SZTE Research Group on Artificial Intelligence, ELRN, Szeged, Hungary
{devroye, mulgund2, rshekh3, gyt, mzefran, yzhou238}@uic.edu

***Abstract*—As new deep-learned error-correcting codes continue to be introduced, it is important to develop tools to interpret the designed codes and understand the training process. Prior work focusing on the deep-learned TurboAE has both interpreted the learned encoders post-hoc by mapping these onto nearby "interpretable" encoders, and experimentally evaluated the performance of these interpretable encoders with various decoders. Here we look at developing tools for interpreting the training process for deep-learned error-correcting codes, focusing on: 1) using the Goldreich-Levin algorithm to quickly interpret the learned encoder; 2) using Fourier coefficients as a tool for understanding the training dynamics and the loss landscape; 3) reformulating the training loss, the binary cross entropy, by relating it to encoder and decoder parameters, and the bit error rate (BER); 4) using these insights to formulate and study a new training procedure. All tools are demonstrated on TurboAE, but are applicable to other deep-learned forward error correcting codes (without feedback).**

I. Introduction

Coding theory aims to develop optimal encoder-decoder pairs for various channels and optimization criteria. This has traditionally been done more or less "by hand" using theoretical insights and mathematical and algorithmic constructions. Recently, however, this has been attempted with a new twist: use machine learning / deep-learning to learn the encoding and/or decoding functions directly [2]–[12]. We refer to such codes as deep-learned error-correcting codes (DL-ECC). The approach has been successful, particularly for channels with feedback [8], [11], [12], but also for point-to-point Additive White Gaussian Noise (AWGN) channels, one of the benchmarks for practical code performance [10], [13].

Deep learning provides computational tools for viewing the task as an optimization problem and solving it efficiently by training a neural network. The rapidly evolving toolkit of deep learning offers many possible architectures and allows (approximately) optimal codes to be found by using a training procedure. For example, for channels with feedback, [8] uses Recurrent Neural Networks, while [12] uses a transformer-based architecture. For point-to-point channels, [10] mimics the architecture of Turbo-codes, replacing convolutional codes with Convolutional Neural Networks (CNN), while [13] uses non-linear learned components in a Reed-Mueller-like construction. Using deep learning and directly optimizing over codes raises important new questions:

1) Can one understand deep learning training procedures in terms of a search process in the space of codes? *We suggest looking at the Fourier expansion of a learned encoder as a way to both understand the final code, and tracking this, as a way to understand the training dynamics. We propose an efficient way to find the dominant Fourier coefficients using the Goldreich-Levin algorithm [14].*

2) If we minimize a loss function over a set of codes (expressible by a given architecture), what can we say about the loss landscape [15]? *We show that for certain architectures and loss functions, parity functions appear to be minimizers.*

3) Loss functions are chosen to facilitate the training process (e.g. binary cross-entropy, BCE), but how do they relate to the performance of a code (e.g. bit error rate, BER)? *We show tight bounds connecting BCE and BER.*

4) If optimization is viewed as working over the two-dimensional (encoder, decoder) space, what can be said about the structure of this space? For example, several training procedures [8], [10] alternate training the encoder and decoder – is this fundamentally needed, or just a practical way to improve convergence? Is there a way to decompose the loss function into an encoder-only and a decoder-only component and exploit this? *We show that there is such a decomposition and suggest an approach that exploits this.*

In this paper, we attempt to study these issues through a mixture of theoretical and experimental insights, focusing on the specific DL-ECC termed TurboAE [10], which has been one of the few DL-ECCs for which interpretability studies have been initiated [16], [17]. Some questions we seek to answer here were inspired by passing remarks in [16] and [17]; we expand upon these. Our experimental observations raise several algorithmic and theoretical questions. This research direction aims to connect coding theory with machine learning interpretability and deep-learning theory.

II. TurboAE: basics and past interpretations

The architecture of the DL-ECC termed "TurboAE" encoder network [10] is based upon a classical rate $\frac{1}{3}$ Turbo code, with

This work was supported by NSF under awards 1900911, 1934915 and 2217023, and by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory (MILAB) program, Hungary. Computing resources for the experiments were provided in part by the NSF award 1828265 (COMPaaS DLV). The authors are in alphabetic order. Proofs and further details are available in [1].

the three "constituent codes" replaced by CNN blocks $f_{b,\theta}(\cdot)$, $b \in \{1,2,3\}$ as in Fig. 1 (adapted from [16]). Similarly, the TurboAE decoder architecture replaces the iterations of the BCJR decoder by CNNs $g_{\phi,1}, g_{\phi,2}$ as in Fig. 1 (adapted from [10]), where $\mathbf{y_b} \in \mathbb{R}^{100}$ (we use bold font for vectors) and $\mathbf{y_b} = \mathbf{x_{AE,b}} + \mathbf{z_b}$, for $\mathbf{z_b}$ i.i.d. Gaussian noise of mean zero and variance 1, for each stream $b \in \{1,2,3\}$. The network is trained in an end-to-end fashion to obtain the network parameters of the encoder and decoder CNNs jointly.

The input to the network is a sequence $\mathbf{u}$ of 100 bits, and the output of each block $b \in \{1,2,3\}$ is a sequence $\mathbf{x}_{AE,b} \in \{\pm 1\}^{100}$. The network has two versions, TurboAE-cont (with real-valued encoder outputs, essentially performing coding and modulation tasks jointly) and TurboAE-binary (with Boolean encoder outputs which are then modulated for transmission over an AWGN channel). The power control modules are omitted, as is the treatment of the boundary first 2 and last 2 bits, discussed in [17] (not needed here).

In [16] "interpretation" of TurboAE-binary was attempted through both exact (non-linear) and approximate (linear, or parity) approximations of the encoding functions $f_{i,\theta}(\cdot)$; these are provided in Tables I and II in the Appendix of [1]. From these, we see that TurboAE-binary's encoders are non-recursive, non-systematic, and $f_{1,\theta}, f_{3,\theta}$ are non-linear and $f_{2,\theta}$ is a linear function of the 5 inputs at times $j-2, \cdots j+2$.

In [16], besides finding the exact and best linear approximations to the encoder functions, several other "interpretation" tools were suggested but not deeply explored. Among these is 1) the use of the Fourier representation of Boolean and pseudo-Boolean functions to better understand the training dynamics, and 2) the suggestion of using the Goldreich-Levin algorithm to find the largest Fourier coefficient(s) as an approximation algorithm for the encoder. We expand on these here, and also look more deeply into the training dynamics, offering an alternative training to that presented in the original TurboAE [10]. All code will be posted on github if the paper is accepted.

III. Interpretations through the Fourier lens

We will later investigate the tracking of Fourier coefficients (FC) of the encoder as a tool for understanding both the training dynamics and the loss landscape. We first discuss an approach to estimate the dominant FC of the learned encoding functions, which may have a large number of inputs.

Changing to the domain $x_i \in \{\pm 1\}$, and letting $\chi_S = \prod_{i \in S} x_i$, each Boolean ($\mathbb{F}_2^n \to \mathbb{F}_2$) and pseudo-Boolean ($\mathbb{F}_2^n \to \mathbb{R}$) function has a unique *Fourier representation* [14]

$$f(x) = \sum_{S \subseteq \{1,2,\cdots,n\}} \hat{f}(S)\chi_S,$$

where $\hat{f}(S)$ is the FC for set S, and $|\hat{f}(S)|^2$ the Fourier weight.

The Goldreich-Levin algorithm (GL) [18] seeks to output a list of sets S for which $|\hat{f}(S)|$ is larger than a pre-specified threshold γ. It requires "query access", which in our context simply means evaluating the neural network on an input.

The theoretical underpinnings are presented in Theorem 1 [14], and our implementation is detailed in the Appendix B of [1]. We explore the applicability of this algorithm in estimating the largest FCs of the encoder of a deep-learned error-correcting code (first proposed for this purpose in [16]), an alternative method to that used in [16] for finding the best parity-approximation. The algorithm has been used in theoretical domains such as cryptography [19], learning theory [20], and coding theory [21], [22] (as a randomized list decoder for Reed-Mueller RM(1,m) codes), but practical implementations and experiments appear limited. We explore some practical aspects of this algorithm.

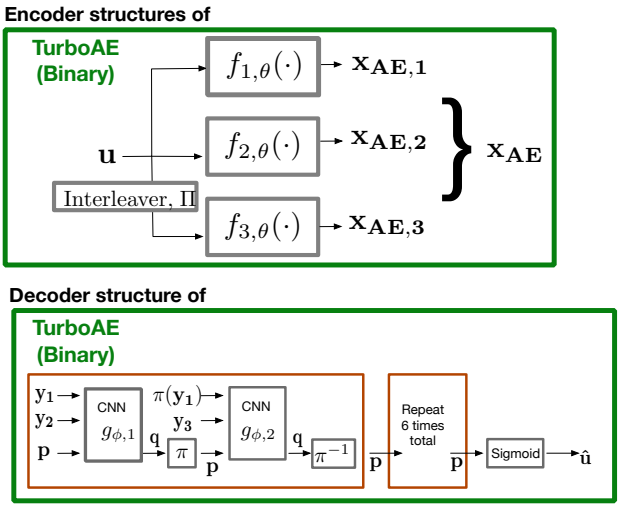


Fig. 1: Above: rate $R = \frac{1}{3}$ ($\mathbf{u} \in \mathbb{F}_2^{100}, \mathbf{x}_{AE,j} \in \{\pm 1\}^{100}$) TurboAE-binary encoder structure. Functions $f_{j,\theta}(\cdot)$ are the constituent codes implemented as CNNs. Below: decoder structure of TurboAE remade from [10]. The parameters of CNNs $g_{\phi,1}, g_{\phi,2}$ are trained. The noisy channel outputs of the three encoded streams $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$ are given by $\mathbf{y_1}, \mathbf{y_2}, \mathbf{y_3}$. The interleaver is π (and its inverse π^{-1}). The decoder produces probabilities that each input bit is 0 or 1, denoted by $\hat{\mathbf{u}}$.

Theorem 1. *[14] Given query access to $f : \{\pm 1\}^n \to \{\pm 1\}$, given $\gamma, \delta > 0$, there is a poly$(n, \frac{1}{\gamma} \log \frac{1}{\delta})$-time algorithm that outputs a list $L = \{S_1, \cdots, S_m\}$ such that (1) if $|\hat{f}(S)| \geq \gamma$, then $S \in L$, and (2) if $S \in L$, then $|\hat{f}(S)| \geq \frac{\gamma}{2}$ holds with probability $1 - \delta$.*

The theorem requires γ to be specified in advance, which holds in several applications. In general, γ can be exponentially small (e.g., for bent functions [23] where every coefficient is $2^{-n/2}$ for n inputs), but it is polynomial, e.g., for functions with small $|L_1|$-norm. If γ is too high then nothing is returned and if it is too low then the running time increases and more coefficients are returned than desired. We are not aware of any work which calculates explicit constants for the number of queries needed in Theorem 1.

We apply GL to TurboAE-binary, exploring its feasibility and the number of queries needed. Previous work [16], [17] is based on the CNN architecture which suggests that the functions in question depend on 9 (TurboAE-cont) or 5 (TurboAE-binary) variables. Here we do not use such a priori knowledge.

Given each TurboAE-binary constituent code $f_{i,\theta} : \{\pm 1\}^{100} \to \{\pm 1\}^{100}$, we randomly select one output bit. We

Block #	Approx. sets for output 11	Min # queries	Approx. $\hat{f}(S)$
1	$\{u_1, u_2, u_3, u_4, u_5\}$	200	-0.81
2	$\{u_1, u_3, u_4, u_5\}$	25	1.0
3	Sol 1: $\{u_1, u_2, u_4\}$ Sol 2: $\{u_1, u_2, u_3, u_4\}$ Sol 3: $\{u_1, u_2, u_4, u_5\}$ Sol 4: $\{u_1, u_2, u_3, u_4, u_5\}$	800	0.4982 -0.5004 -0.5014 -0.4995
where	$u_1 = x_{i+2}$, $u_2 = x_{i+1}$, $u_3 = x_j$, $u_4 = x_{i-1}$, $u_5 = x_{i-2}$, $x =$ inputs		

Fig. 2: Goldreich-Levin approximation for TurboAE-binary.

implemented a heuristic procedure for determining γ for each block and computing the minimum number of queries for each block to get the correct result. Details are given in Appendix B of [1]. How to best pick γ and a minimal number of queries in a principled way is an interesting open question.

Table in Fig. 2 shows the experimental results. The number of queries refers to the number of function evaluations for estimating a single expectation.

IV. Training dynamics: evolution of Fourier coefficients (FC)

Continuing the theme of analyzing Boolean functions in Fourier space, we explore Fourier representation as a tool for understanding the training dynamics and the loss landscape.

A. Dominance and stability of a few Fourier coefficients

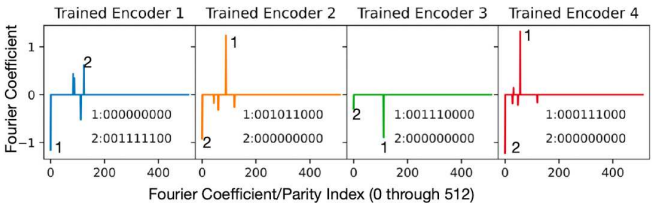
The trained TurboAE was found to have a few dominant Fourier coefficients [16], [17]. One can hypothesize that this might be a general phenomenon when training this network. To investigate this, we trained TurboAE-binary several times from scratch as in [10]. At convergence, the Fourier space appears to almost always be dominated by a few large FCs. In the randomly selected examples in Fig. 3(a) 95% of the total energy (sum of $\hat{f}^2(S)$) is for at most 5 (out of 32) FCs.

Furthermore we observed that at initialization, the dominant FCs almost always correspond to one bit parities. However, with training, higher degree parities emerge as dominant, see Fig. 3(b). Although the setups differ, these observations may be related to recently observed staircase properties [24].

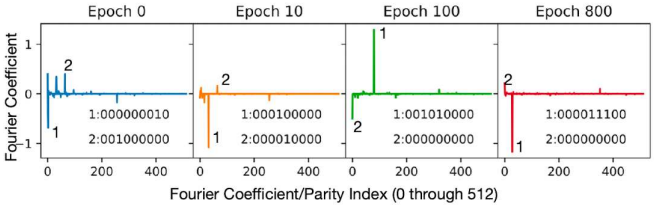
We also trained TurboAE multiple times starting with the same initialization of the neural net weights to evaluate how stable the training process is at convergence. We found that it *is* somewhat stable w.r.t. the dominant Fourier coefficients, as shown for some runs on Fig. 3(c). This begs further questions about the loss landscape of Turbo-like codes, which we propose to study also using a Fourier lens next.

B. Local Optimality of parities for Turbo Codes

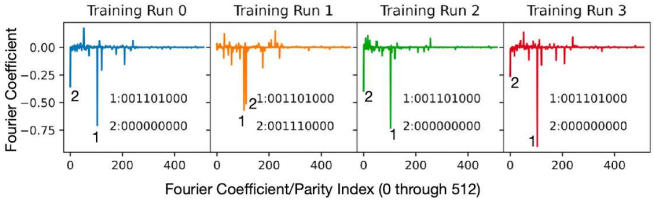
The loss landscape of the TurboAE network is a function over 150,000 parameters, depending on the network. The network parameters of the encoder determine the FCs of the encoding function, and so local minima of the latter can be



(a) Largest FCs of Block 1 with energy $\geq 95\%$ over 4 independent training sessions. Always a few FCs dominate the Fourier space.



(b) FCs of block 3 over different epochs of a training session. Initially, 1-bit parities always dominate, but later higher degree parities emerge.



(c) FCs of block 2 after 4 training sessions with same initialization. The most dominant FC is stable across all runs.

Fig. 3: Progression of TurboAE's Fourier Coefficients with Training. Similar behaviors seen for all the blocks.

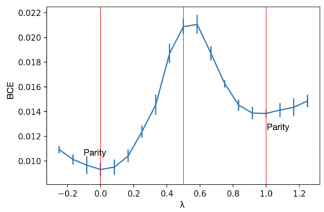
helpful for understanding the former. Thus we study the loss landscape in terms of the Fourier parameterization with 512 parameters, fixing the decoder to BCJR.

To study the loss landscape of generic *non-recursive* Turbo codes in Fourier space, we constructed a parametric Turbo code of block length L(=10), memory 4, parameterized not by CNNs, but by the FC of its constituent codes. It is not clear whether the encoder part of the TurboAE network can implement any triple of 5-variable Boolean functions, therefore the observations might not transfer directly to TurboAE.

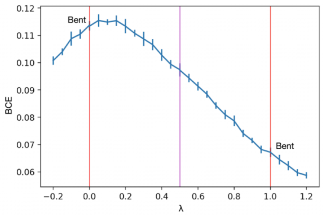
For each block $b \in \{1, 2, 3\}$, use a pseudo-Boolean function $f_{b,\Theta_b} : \{-1, 1\}^5 \to \mathbb{R}$, as the constituent code $f_{b,\theta}(\cdot)$ of Fig. 1, which is completely determined, hence parameterized by its FC $\Theta_b = \hat{f}_b$. These form the Turbo encoder $\mathcal{E}_\Theta$. We use a standard six iteration BCJR decoder for $\mathcal{E}_\Theta$ as the Decoder $\mathcal{D}_\Theta$. We use expected binary cross entropy (BCE) between the input and the decoded output as *loss* $\mathcal{L}$, i.e.

$$\mathcal{L}(\Theta) = \mathop{\mathbb{E}}_{\substack{x\sim\mathcal{U}^L \\ z\sim\mathcal{N}^L}} \left[\frac{1}{L} \sum_{i\in[L]} BCE\left(x_i, \mathcal{D}_\Theta\left(\mathcal{E}_\Theta(x) + z\right)_i\right) \right] \quad .$$

Here z is the *i.i.d.* noise sampled from AWGN channel i.e. $\mathcal{N}_{0,\sigma}$, where σ corresponds to SNR = 1dB. We control the power by keeping the squared sum of the FCs to be 1, which constrains the average power of each bit to also be 1 due to



(a) Line Joining two parities. Local minima at 0 and 1 show that both parities are locally optimal on this line.



(b) Line Joining two non-parity Bent functions. Neither of the two are locally optimal.

Fig. 4: BCE landscape on parametric line $\lambda\Theta'' + (1-\lambda)\Theta'$ joining combinations of parity/non-parity functions. $\lambda = 0, 1$ correspond to Θ', Θ'' respectively.

Parseval's Theorem: $\underbrace{\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}^5}\left[f_b\left(\mathbf{x}\right)^2\right]}_{\text{avg power}} = \sum_{S\subseteq[5]} \hat{f}_b\left(S\right)^2 = \|\Theta_b\|_2^2.$

Our hypothesis is that triples of different parity functions are all local minima, but there are other triples that are not local minima. We ran the following experiment, with results consistent with the hypothesis.

Pick Θ', Θ'' corresponding to triples of different parities $\chi_{\Theta'} = (\chi'_1, \chi'_2, \chi'_3)$ and $\chi_{\Theta''} = (\chi''_1, \chi''_2, \chi''_3)$ respectively. Evaluate $\mathcal{L}$ over several points on the line joining Θ' and Θ'' with power re-normalization. Evaluating a point representing a pseudo-Boolean function involves running BCJR for that function and computing the BCE. We found that Θ' and Θ'' were always local optima on this line. On the other hand, the triple formed by three copies of the bent function $x_1x_2 \oplus x_3x_4$ on different subsets of 5 variables is not a local minimum. The results are illustrated in Fig. 4, and Fig. 8 in the Appendix of [1].

V. Training loss functions: BCE and BER

We now investigate the implications of optimizing BCE from theoretical and empirical perspectives. We consider optimizing an encoder function $f : \mathbb{F}_2^k \to S$ where $S \subset \mathbb{R}^n$ is bounded and $R = k/n$ is our code rate. We take $U \sim \text{Unif}[\mathbb{F}_2^k]$ and $Y \in \mathbb{R}^n$ to be random variables representing the input and received sequence, respectively. Note Y depends on f. Our optimization problem is then:

Problem 1. *Find encoder $f : \mathbb{F}_2^k \to S$ and soft decoder $g \in \mathbb{R}^n \to [0,1]^k$ that minimizes the expected BCE, $\mathbb{C}(f,g)$, where*

$$\mathbb{C}(f,g) = \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} -U_i \lg g_i(Y) - (1-U_i)\lg(1-g_i(Y))\right].$$

A. Theoretical Analysis of BCE Minimization

We re-write the BCE as

$$\mathbb{C}(f,g) = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[D_{KL}(\mathbb{P}[U_i = 1|Y]||g_i(Y))] + \mathbb{H}(U_i|Y) \tag{1}$$

where $\mathbb{H}(U_i|Y)$ is the conditional entropy and $\mathbb{E}[D_{KL}(\mathbb{P}[U_i = 1|Y]||g_i(Y))$ is the expected Kullback-Leibler divergence with the expectation taken over the distribution of received sequences. Note that for a fixed channel, $\mathbb{H}(U_i|Y)$ only depends on the encoder f, and for a fixed encoder the KL-Divergence term $\mathbb{E}[D_{KL}(\mathbb{P}[U_i = 1|Y]||g_i(Y))$ only depends on the decoder, g. We can easily establish the following proposition (proof in Appendix D of [1]):

Proposition 2. *Consider a fixed encoder $f : \mathbb{F}_2^k \to S$. The decoder $g : S \to [0,1]^k$ defined elementwise as $g_i(y) := \mathbb{P}(U_i = 1|Y = y)$ is the unique a.s. minimizer of $\mathbb{C}(f,g)$.*

This is exactly the soft-output MAP decoder, a minimizer of the BER $\mathbb{B}(f,g)$, for a fixed encoder f. This means that, given f, minimizing BCE and BER both reduce to finding a soft-MAP decoder for f. Denote $\mathbb{C}(f), \mathbb{B}(f)$ to be $\mathbb{C}(f, g_{MAP(f)}), \mathbb{B}(f, g_{MAP(f)})$, where $g_{MAP(f)}$ is the soft-MAP decoder for f. Hence, finding a decoder that minimizes BCE finds a decoder that minimizes BER. The same does *not* hold for the general problem of finding an encoder-decoder pair; a memoryless counter-example (asymmetric in the input, unlike the BSC or AWGN) is shown in Appendix F of [1]. The best bounds possible relating BER and BCE are the following, proof in Appendix E of [1].

Proposition 3. *Let $\mathbb{B}_i, \mathbb{C}_i$ denote the BER and BCE respectively on the i^{th} input bit. Then for all choices of f and for all $i \in [k]$, $2\mathbb{B}_i(f) \le \mathbb{C}_i(f) \le \mathbb{H}_2(\mathbb{B}_i(f))$, and in particular*

$$2\mathbb{B}(f) \le \mathbb{C}(f) \le \frac{1}{k}\sum_{i=1}^{k} \mathbb{H}_2(\mathbb{B}_i(f))$$

where $\mathbb{H}_2(p)$ denotes the binary entropy function with parameter $p \in [0,1]$. Furthermore, these bounds are tight in the sense that for any BER $t \in [0, \frac{1}{2}]$ and side of the bound, there exists a channel and an encoder which makes that side an equality.

B. Empirical Application of BCE Decomposition

In light of equation (1), it seems reasonable to decouple the optimization process: optimize the encoder conditional entropy first, then optimize a soft decoder to minimize its KL-divergence with respect to the good encoder. The major obstacle is estimating the conditional entropy of the encoder (or equivalently, the KL-divergence of the decoder). We opt for a naive approach to get around this, but more sophisticated approaches can be found in [25], [26]. If we were optimizing convolutional codes, we could take advantage of the fact that the BCJR [27] is a MAP decoder. However, in the case of general Turbo codes, we have no such efficient MAP decoder. We thus propose training a Turbo-like encoder (top of Fig. 1, but allowing for real-valued outputs) at short block lengths, $k = 16$, using a brute-force marginalization MAP decoder. The encoder is trained with many choices of interleaver so it may generalize well at larger block lengths. Then, we train a neural decoder on the same encoder at our block length $k = 100$. This is in direct contrast to the approach in [10], where the authors alternated between training the encoder and decoder until they jointly converged. See Appendix G of [1] for a more precise formulation of the training procedure.

1) Methods: Like [10], we train a non-systematic, non-recursive turbo code of rate $R = \frac{1}{3}$ for use at block length 100. Rather than training a neural encoder, we directly train the input-output table of a window $w = 5$ (memory 4), possibly nonlinear, automata encoder, borrowing terminology from [28]. We parameterize our encoder by the outputs of a function $h : \mathbb{F}_2^w \to \mathbb{R}^3$. This function is slid over the input bits in $\mathbb{F}_2^k$ as in a non-recursive convolutional encoder. The third stream is convolved over an interleaved copy of the input instead. h is initialized with a normal distribution of mean 0 and variance 1. We also tried initializing h with a parity function, but found it to be at best as good as the normal initialization. Details can be found in the Appendix I of [1].

To enforce the power constraint we use a different method than in [10]. We instead analytically compute power using h. We then center and rescale h after each gradient update so that its power is 1. Precise derivation of this power normalization can be found in the Appendix H of [1]. This helped reduce much of the noise in the training process introduced by the power-normalization in [10]. This method could also be applied when training a neural encoder as in [10]. Once we had a trained encoder, we then trained a neural decoder at larger block length with the same architecture as in [10].

2) Results: The encoder converges fairly quickly (200 steps). The training curve is in [1, Fig. 11, Appendix]. Over several training runs, we were unable to find an encoder with as low a conditional entropy as TurboAE-cont. The discrepancy may come from: (1) our encoder was trained at a block length of 16, while TurboAE-cont was trained at block length 100, and (2) TurboAE-cont was trained by alternating between optimizing the encoder and the decoder, which may have avoided local optima our training scheme runs into.

We show the evolution of the FC for our final encoder in Fig. 5, and the coefficient evolution of another trained encoder in the Appendix of [1]. By the end, only a few dominant FC remain, echoing what we saw in Section IV-A. The FC change significantly from the initialization.

Our decoder optimization proceeds relatively quickly compared to TurboAE-cont; See the [1, Appendix, Fig. 9] for the training curve. The number of steps required for our procedure is only 300,500 whereas TurboAE-cont requires 480,000. In Fig. 6 we see that the performance of our encoder-decoder is slightly worse than TurboAE-cont above SNR 1.0. Observe, if we replace our decoder with BCJR, the performance is almost exactly the same. The decoder may have learned a BCJR-like decoding algorithm which proved to be a local minimum. Nonetheless, our encoder-decoder pair suggests that our training scheme is a viable optimization strategy.

VI. Conclusions

We presented new tools which may help in the interpretation of training aspects of DL-ECCs, including 1) the application of the Goldreich-Levin algorithm to finding the best parity / linear approximation to a black-box encoding function, where its efficiency is useful when there are a large number of input variables but outputs depend only on a few of those

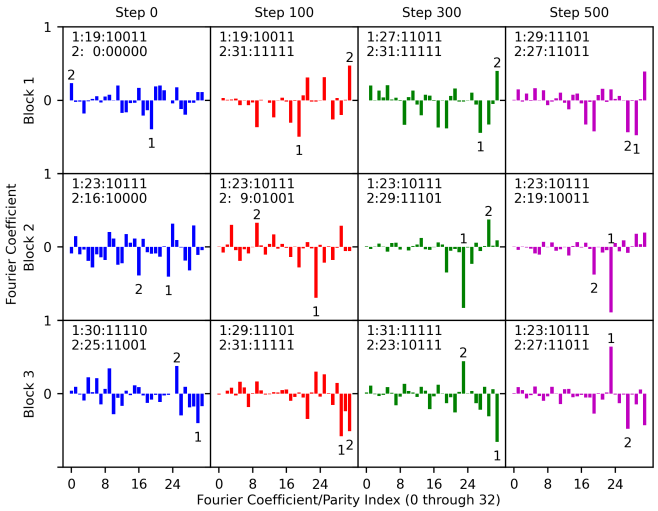


Fig. 5: Evolution of FC during training of the encoder from [1, Fig. 11, Appendix]. The largest coefficients are marked and their corresponding parity is annotated in the respective subplot. Note how a few dominant coefficients emerge and persist during training.

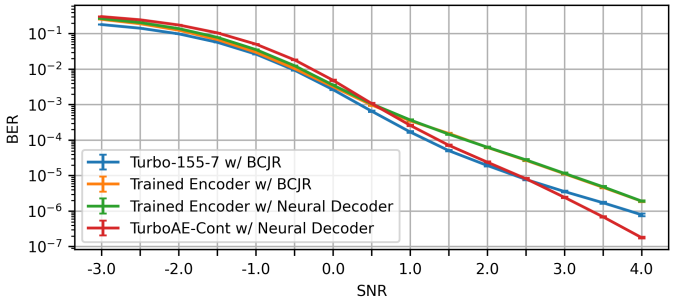


Fig. 6: The BER performance of (1) a benchmark RSC turbo code, (2) our trained encoder paired with BCJR, (3) our trained encoder paired with our neural decoder, (4) TurboAE-cont. TurboAE-cont tends to perform worse at SNRs below 0.5, while showing significant gains at SNRs above 3.0. Our trained neural decoder performs almost the same as BCJR, suggesting it may have learned a similar decoder.

inputs each in the final learned code; 2) the use of FC to understand the loss landscape when training DL-ECCs and also as a possible parameterization for learning codes; 3) observations relating the BCE and BER and a principled alternative approach for training DL-ECCs. While our experiments showed the viability of our alternate training scheme, there are many more aspects to explore: with good estimation of the conditional entropy of an encoder at larger block lengths, we expect performance to improve. In addition, the neural decoder of [10] was designed to mimic an iterative BCJR decoder. However, designing the neural network to mimic exact inference algorithms (e.g. a junction tree [29]) could lead to a better approximation of the MAP decoder. From a bigger-picture perspective, we hope that this decomposition and the usage of Fourier coefficients both as an alternative representation and as a tool for understanding training, will lead to a more principled approach toward the training of DL-ECCs. This could open doors to a more systematic way of finding such codes for different channels.

REFERENCES

[1] N. Devroye *et al.*, "Interpreting Training Aspects of Deep-Learned Error-Correcting Codes – extended ArXiv version," Jun. 2023. [Online]. Available: https://arxiv.org/abs/2305.04347

[2] H. Kim, S. Oh, and P. Viswanath, "Physical Layer Communication via Deep Learning," *IEEE Jour. on Sel. Areas in Info. Theory*, vol. 1, no. 1, pp. 5–18, 2020.

[3] Y. Jiang *et al.*, "Learn codes: Inventing low-latency codes via recurrent neural networks," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 207–216, 2020.

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

[5] Y. Jiang *et al.*, "MIND: Model Independent Neural Decoder," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Jul. 2019, pp. 1–5.

[6] J. Whang *et al.*, "Neural Distributed Source Coding," May 2022. [Online]. Available: http://arxiv.org/abs/2106.02797

[7] R. K. Mishra *et al.*, "Distributed Interference Alignment for K-user Interference Channels via Deep Learning," in *2021 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2021, pp. 2614–2619.

[8] H. Kim *et al.*, "Deepcode: Feedback codes via deep learning," *IEEE Journal on Sel. Areas in Inf. Theory*, vol. 1, no. 1, pp. 194–206, 2020.

[9] Y. Jiang *et al.*, "Joint channel coding and modulation via deep learning," in *SPAWC*, 2020, pp. 1–5.

[10] ——, "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels," in *NIPS*, Dec. 2019, pp. 2758–2768.

[11] K. Chahine, R. Mishra, and H. Kim, "Inventing Codes for Channels with Active Feedback via Deep Learning," *IEEE Journal on Selected Areas in Information Theory*, pp. 1–1, 2022.

[12] E. Ozfatura *et al.*, "All you need is feedback: Communication with block attention feedback codes," *IEEE Jour. on Sel. Areas in Info. Theory*, pp. 1–1, 2022.

[13] A. V. Makkuva *et al.*, "Ko codes: inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning," *ICML*, 2021.

[14] R. O'Donnell, *Analysis of boolean functions*. Cambridge University Press, 2014.

[15] H. Li *et al.*, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[16] N. Devroye *et al.*, "Interpreting Deep-Learned Error-Correcting Codes," in *ISIT*, Jun. 2022, pp. 2457–2462.

[17] ——, "Evaluating interpretations of deep-learned error-correcting codes," in *Allerton*, Sep. 2022.

[18] O. Goldreich and L. A. Levin, "A hard-core predicate for all one-way functions," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 1989, pp. 25–32.

[19] J. Håstad *et al.*, "A pseudorandom generator from any one-way function," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.

[20] E. Kushilevitz and Y. Mansour, "Learning decision trees using the fourier spectrum," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 1991, pp. 455–464.

[21] A. Akavia, S. Goldwasser, and S. Safra, "Proving hard-core predicates using list decoding," in *FOCS*, vol. 44. Citeseer, 2003, pp. 146–159.

[22] A. S. Abdouli *et al.*, "The Goldreich-Levin algorithm with reduced complexity," in *Thirteenth International Workshop on Algebraic and Combinatorial Coding Theory (ACCT'12)*, Pomorie, Bulgaria, Jun. 2012, pp. 7–14.

[23] O. S. Rothaus, "On "bent" functions," *Journal of Combinatorial Theory, Series A*, vol. 20, no. 3, pp. 300–305, May 1976.

[24] E. Abbe *et al.*, "The staircase property: How hierarchical structure can guide deep learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 26 989–27 002.

[25] B. Poczos and J. Schneider, "Nonparametric Estimation of Conditional Information and Divergences," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, 2012, pp. 914–923.

[26] L. Paninski, "Estimation of entropy and mutual information," *Neural Computation*, vol. 15, no. 6, p. 1191–1253, Jun 2003.

[27] L. Bahl *et al.*, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[28] L. Bazzi, M. Mahdian, and D. A. Spielman, "The minimum distance of turbo-like codes," *IEEE Transactions on Information Theory*, vol. 55, no. 1, p. 6–15, Jan 2009.

[29] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*, ser. Monographs in Computer Science. New York, NY: Springer, 1997.