Iterative Error Decimation for Syndrome-Based Neural Network Decoders

Jorge Kysnney Santos Kamassury and Danilo Silva

***Abstract*—In this letter, we introduce a new syndrome-based decoder where a deep neural network (DNN) estimates the error pattern from the reliability and syndrome of the received vector. The proposed algorithm works by iteratively selecting the most confident positions to be the error bits of the error pattern, updating the vector received when a new position of the error pattern is selected. Simulation results for the (63,45) and (63,36) BCH codes show that the proposed approach outperforms existing neural network decoders. In addition, the new decoder is flexible in that it can be applied on top of any existing syndrome-based DNN decoder without retraining.**

***Index Terms*—Short-Length Codes, Syndrome, Iterative Error Decimation, Deep Neural Network, BCH.**

I. Introduction

In recent years, investigations into the design of short-length channel codes have acquired notability, particularly due to applications that newer technologies aim to support. 5G technology, in particular, aims to guarantee services that require ultra-reliable low-latency communication (URLLC) [1]. For example, intelligent transport systems and process automation demand reliability in the order of 10^{-3} to 10^{-6} and latency between 1 ms to 100 ms. Communication under these conditions is challenging, since the requirements themselves are strict and conflicting [2], [3].

This scenario has motivated the evaluation of possible candidate codes in terms of reliability and complexity for a given (short) blocklength [2], [4], [5]. Among many candidates—which include polar, LDPC and convolutional codes—BCH codes stand out as having an excellent performance, very close to the fundamental limits in the short blocklength regime. This is achieved by the use of an ordered statistics decoder (OSD), which delivers near-maximum-likelihood (ML) performance; however, this comes at the price of a high complexity, which grows quickly as the blocklength increases.

An alternative that has increasingly been explored in recent work is the use of decoders based on deep neural networks (DNNs). Although the use of neural networks (NNs) for the task of decoding is not recent [6], due to the success of deep learning in several applications, interest in this purpose has been resumed [7]. Recently, in [8], Nachmani *et al.* proposed a deep learning framework that is modeled on the LDPC belief propagation (BP) decoder, where connections between neurons (as well as activations) are designed to mimic the underlying Tanner graph. In subsequent works [9]–[12], other architectures based on [8] are presented.

Unlike approaches based on BP decoding, Bennatan *et al.* proposed in [13] a new decoder structure, where the NN is fed the reliability and syndromes of the received sequences and acts on noise estimation. Their approach can be regarded as a soft-decision extension of the syndrome-based approach of [6]. A great advantage of this structure is that the NN can be designed freely, i.e., without the restrictions present in architectures based on the BP decoder. Subsequently, the vanilla DNN proposed in [13] was simplified in [14], [15]; specifically, the architecture in [15] has fewer parameters and achieves a better performance than the original one.

A common limitation in many previous works is their focus on the bit error rate (BER) as a measure of performance, presumably because it maps more directly to the NN training objective. However, when evaluated by the block error rate (BLER), some of these works fail to significantly improve upon a hard-decision bounded-distance decoder (HD-BDD) that would conventionally be used to decode BCH codes.

In this paper, we present a strategy to improve the performance of any syndrome-based neural decoder (i.e., any decoder following the approach in [13]), at the expense of a moderate increase in complexity. Our approach is to take the unquantized estimate of the error vector that is output by a NN decoder and iteratively select its most confident position, which is then *decimated* (subtracted) from the received vector before a new decoding attempt is made. Our results show that this proposed approach significantly improves the BLER achieved by the decoder in [13], outperforming previous results for the BCH(63,36) and BCH(63,45) codes.[1]

Notation: We use x_i for the ith element of a vector $\mathbf{x}$. Let $\mathbf{0}$ and $\mathbf{1}$ be the all-zeros and the all-ones vectors, respectively, with lengths implied by the context. If $\mathbf{x} \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$, then $1[\mathbf{x} > \gamma]$ denotes the vector $\mathbf{y} \in \{0,1\}^n$ such that $y_i = 1$ if and only if $x_i > \gamma$. We use a similar notation for $1[\mathbf{x} < \gamma]$.

II. Preliminaries

A. Channel model

Let $C \subseteq \{0,1\}^n$ be an (n,k) binary linear code with parity-check matrix $\mathbf{H} \in \{0,1\}^{(n-k)\times n}$. Suppose a codeword $\mathbf{c} \in C$ chosen uniformly at random is transmitted over a binary-input additive white Gaussian noise (BI-AWGN) channel. The received vector is given by

$$\mathbf{y} = \mathbf{1} - 2\mathbf{c} + \mathbf{z} \tag{1}$$

J. K. S. Kamassury and D. Silva are with the Department of Electrical and Electronic Engineering, Federal University of Santa Catarina, Florianópolis-SC, Brazil (e-mail: jorge.kamassury@posgrad.ufsc.br; danilo.silva@ufsc.br).

This work was partially supported by CNPq under Grants 132881/2018-7, 310343/2016-0, 429097/2016-6 and 309413/2019-2.

Digital Object Identifier: 10.14209/jcis.2021.16

[1] Code available at https://github.com/Kamassury/IED.

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_n)$ and $\sigma^2 = N_0/(2E_b)$. The goal of the decoder is to infer $\mathbf{c}$ from $\mathbf{y}$, producing an estimate $\hat{\mathbf{c}} \in \{0,1\}^n$. The block error probability (BLER) is defined as $P[\hat{\mathbf{c}} \neq \mathbf{c}]$.

B. Syndrome-Based Neural Decoding

Let $\mathbf{y}_b = 1[\mathbf{y} < 0] \in \{0,1\}^n$ be the vector of hard decisions[2] and let $\mathbf{e} = \mathbf{y}_b + \mathbf{c} \bmod 2 \in \{0,1\}^n$ be corresponding error vector. Clearly, $\mathbf{c}$ can be easily found given $\mathbf{y}_b$ and $\mathbf{e}$. Thus, the decoding problem reduces to that of estimating $\mathbf{e}$. As shown in [13], a sufficient statistic for the estimation of $\mathbf{e}$ is the pair $(\mathbf{s}, |\mathbf{y}|)$, where $\mathbf{s} = \mathbf{y}_b \mathbf{H}^T \bmod 2$ is the *syndrome* of the error vector (i.e., $\mathbf{s} = \mathbf{e}\mathbf{H}^T \bmod 2$) and $|\mathbf{y}| = (|y_1|, \ldots, |y_n|)$ is the vector of channel reliabilities.

The approach proposed in [13] is to design an NN to estimate $\mathbf{e}$ from $(\mathbf{s}, |\mathbf{y}|)$. More precisely, the network is trained to minimize the empirical risk $E[\sum_{i=1}^n L(e_i, \tilde{e}_i)]$ under the channel distribution, where $L(e_i, \tilde{e}_i) = -e_i \log \tilde{e}_i - (1-e_i)\log(1-\tilde{e}_i)$ is the binary cross-entropy (BCE) loss function and $\tilde{\mathbf{e}} \in [0,1]^n$ is the NN output, produced with a sigmoid output activation function.[3] The binary estimate of $\mathbf{e}$ is then obtained as $\hat{\mathbf{e}} = 1[\tilde{\mathbf{e}} > 0.5] \in \{0,1\}^n$. The complete decoder, which we refer to as a syndrome-based neural decoder (SBND), is shown in Fig. 1.

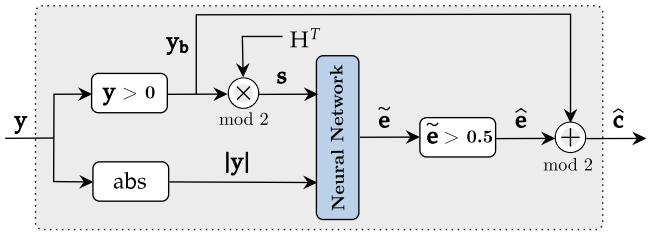


Fig. 1: A general syndrome-based neural decoder.

As argued in [13], the inputs $(\mathbf{s}, |\mathbf{y}|)$ and the target $\mathbf{e}$ are all independent of $\mathbf{c}$, thus the zero codeword assumption $\mathbf{c} = \mathbf{0}$ can be used for both training and performance evaluation of the decoder. This avoids the risk of overfitting to the subset of codewords used during training. Moreover, as with any neural decoder, since the channel model is known, a potentially unlimited number examples can be used for training and testing without risk of overfitting to the noise.

III. Iterative Error Decimation Decoder

A. Motivation

A main issue in training a syndrome-based neural decoder according to the procedure in Section II-B is the potential presence of inconsistent (or "noisy") training examples, namely, training examples with the same (or very similar) inputs but different targets. This phenomenon, called *disturbance* in [6], is most clearly seen in a decoder where the input component $|\mathbf{y}|$ is removed from the neural network, i.e., the neural network is trained to predict the target error vector $\mathbf{e}$ solely from its syndrome $\mathbf{s}$. Note that this corresponds to degrading the BI-AWGN channel into a binary symmetric channel (BSC), which is the channel originally considered in [6]. In this case, multiple target error vectors with the same syndrome are likely to appear during training, producing a "noisy" output that tends to be a superposition of those error vectors.

For simplicity, consider the BSC case in the following. Ideally, the neural network should be trained to emulate the performance of a maximum-likelihood decoder; thus, every syndrome $\mathbf{s}$ should be paired with a *single* lowest-weight error vector $\mathbf{e}$ corresponding to that syndrome, in order to form the training set. Any distinct error vector with the same syndrome, if used as a training example, will drive the network to deviate from the desired prediction and thus can only hurt performance. However, generating such an optimal training set requires performing maximum-likelihood decoding for every possible syndrome (or, equivalently, generating and storing a full syndrome table) which can be computationally infeasible.

A simple approach proposed in [6] to avoid disturbance is to restrict the training set to only target error vectors of weight up to the guaranteed error-correction capability of the code, $t = \lfloor (d_{\min}-1)/2 \rfloor$, where $d_{\min}$ is the minimum distance of the code. This set is guaranteed to have a single error vector for each syndrome. However, under this approach, the neural network is unlikely to learn to predict error vectors of larger weights, which is precisely what is needed in order to outperform a bounded-distance decoder.

Now, let us illustrate what can happen when an inconsistent training set is used. For instance, consider a $(15, 5, 7)$ BCH code. For this code, the error vectors

$$\mathbf{e}_1 = (1,1,1,1,0,0,0,0,0,0,0,0,0,0,0)$$
$$\mathbf{e}_2 = (0,0,0,0,0,1,0,1,1,0,0,1,0,0,0)$$

have exactly the same syndrome (and these are the only lowest-weight vectors with that syndrome). For that syndrome, the output of an NN (trained as in Section II-B) may be, e.g.,

$$\tilde{\mathbf{e}} = (0.479, 0.505, 0.512, 0.491, 0.005, 0.507, 0.000, 0.516, 0.481, 0.000, 0.000, 0.483, 0.002, 0.001, 0.000)$$

which, after thresholding at 0.5, leads to the estimate

$$\hat{\mathbf{e}} = (0,1,1,0,0,1,0,1,0,0,0,0,0,0,0).$$

This prediction is always incorrect, as it does not even correspond to the input syndrome.

An explanation for this behavior is that, under the architecture and training approach of Section II-B, the NN is modeling the *bitwise* posterior probability

$$\tilde{e}_j \approx P[e_j = 1 \mid (\mathbf{s}, |\mathbf{y}|)].$$

While this approach can potentially lead to a low bit-error rate (BER), it is clearly unsuited to obtaining low BLER. On the other hand, regarding the problem as a multiclass classification among all possible error vectors (e.g., using softmax output activation with categorical cross-entropy loss) [16] is clearly computationally infeasible unless n is very small.

[2]Note that $y_i = -1 + z_i$ when $c_i = 1$.

[3]The original description in [13] uses a $[-1, 1]$ mapping and a hyperbolic tangent output activation function, which is mathematically equivalent to the description given here.

B. Iterative Error Decimation

Rather than modifying the training procedure to avoid disturbance as in [6], we propose to modify the decoder so as to make it robust to the superposition of error patterns.

Our approach is to perform $T-1$ iterations where a *single* bit is selected that is most likely (as estimated by the neural network) to be in error; this bit is then flipped in the received vector and the decoding is repeated, until the Tth iteration where thresholding at 0.5 is applied. We call this procedure *iterative error decimation* (IED). The underlying idea is that, after a bit error is (correctly) eliminated, the resulting problem becomes easier to solve, leading to more confident estimates. Note that IED can be applied to any syndrome-based neural decoder, without requiring any changes in the training stage.

A detailed description of the decoder is given in Algorithm 1. Note that, in line 8, we assume that the NN outputs probability estimates. In line 10, we select the position j of the largest (thus, most confident) element of the vector $\tilde{\mathbf{e}}$. The decimation step occurs at line 11, where the sign of the received vector is flipped at the position j estimated to be in error. Since we assume certainty that the chosen position is in error, in principle we could also set the magnitude $|\mathbf{y}_j|$ to infinity (or to a very large value). However, in our experiments we observed that setting $|\mathbf{y}_j|$ to a too high value actually hurts performance, possibly because such values were not observed during training. In practice, we found that the best results are obtained when we do not change the magnitude of $|\mathbf{y}_j|$.

The algorithm stops when a zero syndrome has been obtained (line 4) or when T iterations have been performed, at which point thresholding is applied to the remaining error estimate.

Clearly, the complexity of one iteration of the IED decoder is dominated by that of the NN inference step. Since the number of iterations is at most T, the maximum latency is at most T times that of a conventional SBND. On the other hand, the average number of iterations is upper bounded by

$$1 + P[\mathcal{E}_1] + \cdots + P[\mathcal{E}_1, ..., \mathcal{E}_{T-1}] \leq 1 + P[\mathcal{E}_1] + ... + P[\mathcal{E}_{T-1}]$$

where $P[\mathcal{E}_i]$ is the block error probability of an IED decoder with i iterations. Thus, compared to a conventional SBND, the relative increase in the average complexity is typically very small and becomes negligible for high E_b/N_0.

Algorithm 1 Iterative error decimation (IED) decoder

Input: $\mathbf{y}$, H, T
Output: $\hat{\mathbf{c}}$
1: **for** $i = 1, \ldots,$ T **do**
2: $\quad \mathbf{y_b} \leftarrow 1[\mathbf{y} < 0]$
3: $\quad \mathbf{s} \leftarrow \mathbf{y_b}\mathrm{H}^T \bmod 2$
4: $\quad$ **if** $\mathbf{s} = 0$ **then**
5: $\quad\quad \hat{\mathbf{c}} \leftarrow \mathbf{y_b}$
6: $\quad\quad$ **return** $\hat{\mathbf{c}}$
7: $\quad$ **end if**
8: $\quad \tilde{\mathbf{e}} \leftarrow \mathrm{NN}(\mathbf{s}, |\mathbf{y}|)$
9: $\quad$ **if** $i <$ T **then**
10: $\quad\quad j \leftarrow \arg\max(\tilde{\mathbf{e}})$
11: $\quad\quad \mathbf{y}_j \leftarrow -\mathbf{y}_j$
12: $\quad$ **end if**
13: **end for**
14: $\hat{\mathbf{e}} \leftarrow 1[\tilde{\mathbf{e}} > 0.5]$
15: $\hat{\mathbf{c}} \leftarrow \mathbf{y_b} + \hat{\mathbf{e}} \bmod 2$
16: **return** $\hat{\mathbf{c}}$

IV. Experiments and results

In this section, we investigate the BLER performance of the decoders described in the sections II-B and III-B for the linear codes BCH(63,45) and BCH(63,36), where BCH(n, k) denotes a primitive narrow-sense binary BCH code of length n and dimension k. For comparison purposes, we use the best results obtained in [11], [12], [15] as well as the HD-BDD and ML [17] performances. With respect to BER performance, we compare specifically with [12], [15] and [18] (note that [11] presents only BLER performance). All simulations were performed using the Keras API with Tensorflow backend.

For the training of DNNs, we have used 10^7 examples (generated in real time) with $E_b/N_0 = 4$ dB. This value of E_b/N_0 is suggested in [7] to give a good balance between noise and code structure in the training examples presented for the DNN to learn. We have used Glorot normal initialization and the Adam optimizer with batches of size 2048.

In the inference stage, the BLER was estimated by running Monte Carlo simulations until the occurrence of at least 100 block errors for each value E_b/N_0.

A. BCH(63,45) code

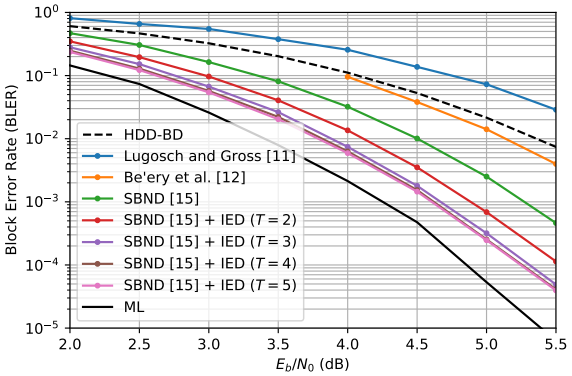
For the BCH(63,45) code, we use the DNN architecture presented in [15], which has seven fully connected layers. The six hidden layers have 300 units each and use a rectified linear unit (ReLU) as activation function [19].

Following the same procedures described in [15], for this architeture the learning rate for the gradient propagation is initialized to 10^{-3} and is reduced by a factor of 10^{-1} when the validation loss stops reducing for 5 epochs.

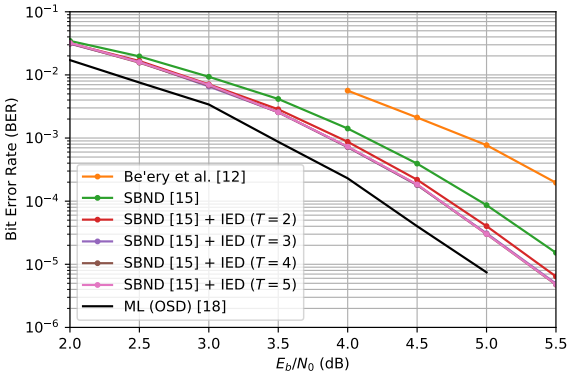
Fig. 2 shows the performance achieved with the SBND proposed in [13] and the IED decoder using the DNN designed in [15]. It is observed that the result obtained in [15] already exceeds the performances shown in [11], [12]. In turn, with the same DNN and using the proposed IED decoder we achieve even better performance. For the interval $T \in [2, 5]$, we observe a gradual improvement, reaching up to 0.7 dB (for $T = 5$) compared to the result obtained in [15], when BLER = 10^{-3}. Our tests indicate that, for $T > 5$, the improvement is not significant.

B. BCH(63,36) code

For the BCH(63,36) code, we propose the 8-layer architecture, with seven fully connected hidden layers, each of which has $8n = 504$ units and uses the logistic sigmoid activation function. We also include a single skip connection (concatenation) from the first to the fourth layer. All hidden layers are followed by batch normalization layers to help with the stability and acceleration of the learning process [19].

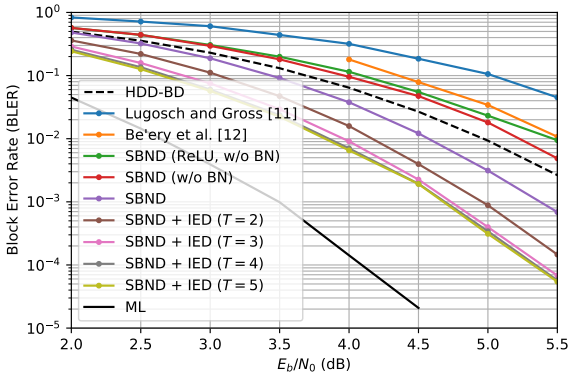


(a)

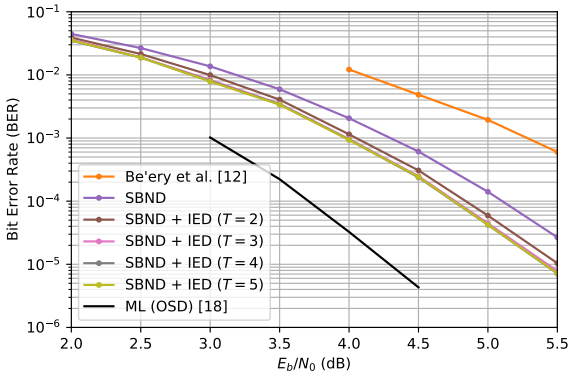


(b)

Fig. 2: Performance obtained with the decoder in [13] and the IED decoder for the BCH(63,45) code, using the DNN in [15].



(a)



(b)

Fig. 3: Performance obtained with the decoder in [13] and the IED decoder for the BCH(63,36) code using the DNN proposed in section IV-B.

For the learning rate, we obtained our best results by applying a triangular cyclic schedule [20] with minimum at 10^{-5}, maximum at 10^{-3}, and a half-cycle of 64 iterations.

Fig. 3 shows the performance of the proposed DNN with the decoder in [13] and the IED decoder. In Fig. 3(a), "(w/o BN)" indicates a version with the batch normalization layers removed and "(relu, w/o BN)" indicates a further modification where the sigmoid activation of the hidden layers is replaced by ReLU. We can see that the combined use of the sigmoid activation and the batch normalization layers significantly improves the performance.

Again, it can be seen that the IED decoder achieves better performance than the results in the literature, including those of [13]. As in the case of the BCH(63,45) code, our best result is obtained when $T = 5$, providing a gain of approximately 0.8 dB at BLER = 10^{-3}.

C. Comparison with the Syndrome Loss

To investigate whether the problem of disturbance discussed in Section III-A could be solved by simply penalizing syndrome violations (without IED), we have trained the DNNs of sections IV-A and IV-B using the decoder of [13] and the hybrid loss function proposed in [11], which incorporates a syndrome loss component besides the BCE loss. We experimented training from scratch and after pretrainng with the BCE loss. However, in both cases, the results were worse than using only the BCE loss and therefore were not included in the figures. This is not surprising since the syndrome loss was proposed in the context of belief-propagation decoding. Moreover, it ideally implies committing to a single rather than multiple superimposed error vectors, which may simply be too hard to learn under an inconsistent training set. In contrast, the BCE loss makes no such commitment, allowing the first iteration of the IED decoder to find and flip the single bit that is most likely to be in error.

V. Conclusion

In this letter, we proposed a new decoder that uses the knowledge of the syndrome vector to feed a DNN designed to estimate the error pattern, where a stage of selecting the most confident positions to correspond to errors is used in order to improve estimation of the transmitted codeword. In addition, we designed a new DNN for decoding the BCH(63,36) code.

The results obtained for the BCH(63,45) and BCH(63,36) codes show that the new decoding algorithm improves the performance of the SBND presented in [13], at the price of a moderate increase in complexity. The IED decoder is flexible in the sense that it can be directly applied to any syndrome-based neural decoder without retraining.

REFERENCES

[1] V. K. Huang, Z. Pang, C.-J. A. Chen, and K. F. Tsang, "New trends in the practical deployment of industrial wireless: From noncritical to critical use cases," *IEEE Industrial Electronics Magazine*, vol. 12, no. 2, pp. 50–58, 2018, doi: 10.1109/MIE.2018.2825480.

[2] M. Shirvanimoghaddam, M. S. Mohammadi, R. Abbas, A. Minja, C. Yue, B. Matuz, G. Han, Z. Lin, W. Liu, Y. Li, S. Johnson, and B. Vucetic, "Short block-length codes for ultra-reliable low latency communications," *IEEE Communications Magazine*, vol. 57, no. 2, pp. 130–137, 2019, doi: 10.1109/MCOM.2018.1800181.

[3] G. Durisi, T. Koch, and P. Popovski, "Toward massive, ultrareliable, and low-latency wireless communication with short packets," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1711–1726, 2016, doi: 10.1109/JPROC.2016.2537298.

[4] G. Liva, L. Gaudio, T. Ninacs, and T. Jerkovits, "Code Design for Short Blocks: A Survey," *arXiv:1610.00873 [cs, math]*, Oct. 2016. [Online]. Available: https://arxiv.org/abs/1610.00873

[5] M. C. Coşkun, G. Durisi, T. Jerkovits, G. Liva, W. Ryan, B. Stein, and F. Steiner, "Efficient error-correcting codes in the short blocklength regime," *Physical Communication*, vol. 34, pp. 66–79, Jun. 2019, doi: 10.1016/j.phycom.2019.03.004.

[6] L. G. Tallini and P. Cull, "Neural nets for decoding error-correcting codes," in *IEEE Technical Applications Conference and Workshops. Northcon/95. Conference Record*, 1995, pp. 89–, doi: 10.1109/NORTHC.1995.485019.

[7] T. Gruber, S. Cammerer, J. Hoydis, and S. t. Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, 2017, pp. 1–6, doi: 10.1109/CISS.2017.7926071.

[8] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2016, pp. 341–346, doi: 10.1109/ALLERTON.2016.7852251.

[9] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018, doi: 10.1109/JSTSP.2017.2788405.

[10] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 1361–1365, doi: 10.1109/ISIT.2017.8006751.

[11] ——, "Learning from the syndrome," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 594–598, doi: 10.1109/ACSSC.2018.8645388.

[12] I. Be'Ery, N. Raviv, T. Raviv, and Y. Be'Ery, "Active deep decoding of linear codes," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 728–736, 2020, doi: 10.1109/TCOMM.2019.2955724.

[13] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes - a syndrome-based approach," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1595–1599, doi: 10.1109/ISIT.2018.8437530.

[14] E. Kavvousanos, V. Paliouras, and I. Kouretas, "Simplified deep-learning-based decoders for linear block codes," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018, pp. 769–772, doi: 10.1109/ICECS.2018.8617843.

[15] E. Kavvousanos and V. Paliouras, "Hardware implementation aspects of a syndrome-based neural network decoder for bch codes," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2019, pp. 1–6, doi: 10.1109/NORCHIP.2019.8906946.

[16] C. T. Leung, R. V. Bhat, and M. Motani, "Low-latency neural decoders for linear and non-linear block codes," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6, doi: 10.1109/GLOBECOM38437.2019.9014156.

[17] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of Channel Codes and ML Simulation Results," www.uni-kl.de/channel-codes, 2019.

[18] E. Nachmani, Y. Bachar, E. Marciano, D. Burshtein, and Y. Be'ery, "Near maximum likelihood decoding with deep learning," in *International Zurich Seminar on Information and Communication (IZS 2018)*, 2018, pp. 40–44, doi: 10.3929/ethz-b-000245051.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[20] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017, pp. 464–472, doi: 10.1109/WACV.2017.58.