2021 XVII International Symposium Problems of Redundancy in Information and Control Systems (REDUNDANCY) | 978-1-6654-3308-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/REDUNDANCY52534.2021.9606471

Deep Neural Network Based Decoding of Short 5G LDPC Codes

Kirill Andreev*, Alexey Frolov*, German Svistunov †, Kedi Wu †, Jing Liang ‡
* Skolkovo Institute of Science and Technology, Moscow, Russian Federation
† Huawei Technologies Co. Ltd, Moscow, Russian Federation
‡ Huawei Technologies Co. Ltd, Department of R&D
k.andreev@skoltech.ru, al.frolov@skoltech.ru, svistunov.german1@huawei.com, wukedi@huawei.com, jingliang@huawei.com

***Abstract*—We investigate the application of machine learning techniques (in particular, deep neural networks, DNN) to improve the decoding algorithms of short quasi-cyclic low-density parity-check (LDPC) codes adopted to the 5G standard. We note that straightforward application of general-purpose DNNs is not possible due to the curse of dimensionality problem – the training set size grows exponentially with the number of information bits. In our opinion, the only way to deal with this problem is to combine deep learning methods with existing decoding algorithms. We start with a Tanner-based neural network decoder with Min-Sum activation functions proposed by Nachmani et al. and extend it as follows. First, the quasi-cyclic nature of 5G LDPC codes allows us to use a single weight per circular matrix (circulant). We refer to this as weight sharing. This idea significantly reduces the training time, preserving the error-correcting performance. Second, we add residual connections to our NN architecture. Residual connections improve the performance and reduce the training time. We also present the results for the rate and length adaptation techniques. Rate adaptation allows multiple DNNs corresponding to different coding rates to run with a single set of trained weights. Length adaptation allows optimally reusing weights for multiple lifting size indices.**

I. Introduction

The 5G standardization has adopted the multi-edge raptor-like LDPC [1, 2, 3, 4] codes for enhanced mobile broadband traffic scenario (eMBB). LDPC codes have excellent performance for moderate and large lengths but are less efficient for short code-lengths due to small trapping sets. In this paper, we address this question and investigate the application of deep learning methods to improve the decoding algorithms of short 5G LDPC codes. This results in improved coverage, which is of critical importance for 5G Internet of Things (IoT) applications. A good-performing LDPC decoder allows us to improve the base station coverage, which is of critical importance for 5G Internet of Things (IoT) applications.

The idea to use NNs in the channel decoding problem is not a new one [5, 6, 7, 8]. It is easy to see that the decoding task is a classification task: the channel output must correspond to one of the classes (codewords). The significant difference between this problem and a typical classification problem lies in the exponentially large number of classes. The straightforward approach consists of the NN having multiple dense layers. But this solution suffers from the curse of dimensionality: the training set size grows exponentially with the number of information bits. This number is too large to fully train the NN (in other words, show it all the codewords). The only hope is that the NN can learn the code structure by seeing only a few codewords. Unfortunately, a general-purpose NN does not have a generalization ability and can not learn the code structure by observing a small number of code words [9]. Thus, this approach can only work in the case of a small number of codewords.

In our opinion, the only way to deal with the curse of dimensionality problem is to combine deep learning methods with existing decoding approaches. We note the papers utilizing belief propagation decoding algorithms (Sum-Product [5] or Min-Sum [10]), partitioning approach [7], and syndrome decoding [8]. The authors of these papers have focused their efforts on the soft decoding of BCH codes using NNs. Unfortunately, they did not improve the decoding performance compared to the best soft BCH decoders (for example, the Chase method [11] or an ordered statistics decoding [12]). The explanation is quite simple: the approaches listed above do not work well for BCH codes, and even using machine learning methods does not solve this problem.

The authors of [13] proposed a recurrent neural network (RNN) to decode BCH codes. Weight sharing [14] is a way to combine RNN with deep architecture when some set of weights take participation in multiple decoding iterations. We note that our weight sharing approach discussed below uses a quasi-cyclic structure of LDPC codes, resulting in a single trainable weight per circulant rather than a single weight per multiple decoding iterations. The authors of [15] present the message passing algorithm with quantized messages and the trainable weights. There are also some attempts to improve the architecture of the Tanner-based NN [5]. The hypernet approach presented in [16] is such an example. Then, [17] investigates the hypernet-based iterative polar decoder. Another application of machine learning to the channel decoding problem is [18], where the authors suggested automatic code structure learning using deep neural networks.

In this paper, we focus on binary LDPC codes. Our goal is to improve the performance of short 5G LDPC codes using machine learning methods. As belief propagation decoding is natural for LDPC codes, we start with a Tanner-based NN decoder with Min-Sum activations proposed in [5].

Our contribution is the following. First, the quasi-cyclic nature of 5G LDPC codes allows us to use weight sharing.

Weight sharing is the use of a single weight per circular matrix (circulant). Second, we add residual connections to our NN architecture. Both ideas reduce the training time and improve the decoding performance. As a result, a neural Min-Sum decoder achieves the same performance as a Sum-Product decoder with the same number of iterations. Moreover, for some LDPC constructions, we have achieved a performance gain of about 0.3 dB. And third, rate and length adaptation. The length adaptation allows optimal weight reuse, while the rate adaptation allows covering a wide range of coding rates with a small number of optimal weight sets. In our simulation setup, we have covered 23 coding rates with just three sets of optimal weights.

II. Neural network architecture

The neural network (NN) architecture is a Tanner graph-based decoder with embedded trainable weights [5]. The main principle behind this NN is a belief-propagation decoding algorithm[1]. The following subsections describe the belief propagation algorithm and corresponding NN architecture.

A. Belief propagation iterative decoder

The belief propagation algorithm is the message passing algorithm, with the messages being some functions of probabilities (e.g., LLRs). One can represent any linear code via a bipartite Tanner graph [19]. The nodes of the first type (variable nodes) correspond to coded bits. The second node types (check nodes) correspond to linear parity checks. There is an edge between some variable and check node if the corresponding bit takes participation in the parity check (see Fig. 1).

Let the $Q_{i\to j}$, $j = 1, \ldots, N$, where N is the codelength be the messages from the variable to check nodes and $R_{i\to j}$ – from check to variable nodes. The decoder is initialized with $Q_{i\to j} = L_i, i = 1, \ldots, M$, where M is the number of parity checks. Each iteration of this iterative decoding algorithm consists of two steps: check node operations and variable node operations.

a) Check nodes operations:

$$R_{j\to i} = 2 \operatorname{arctanh} \left(\prod_{i' \in \Gamma(j), i' \neq i} \tanh \left(\frac{Q_{i'\to j}}{2} \right) \right), \quad (1)$$

where $\Gamma(j)$ is the set of variable nodes adjacent to given check node. This optimal rule (called the Sum-Product or SP) is not used in practice due to numerical complexity in fixed-point implementations. The Min-Sum algorithm (MS) is the SP approximation:

$$R_{j\to i} = \prod_{i' \in \Gamma(j), i' \neq i} \operatorname{sign}(Q_{i'\to j}) \min_{i' \in \Gamma(j), i' \neq i} |Q_{i'\to j}|. \quad (2)$$

b) variable nodes operations: The variable node operations are the same for MS and SP algorithms.

$$Q_{i\to j} = L_i + \sum_{j' \in \Gamma(i), j' \neq j} R_{j'\to i}, \quad (3)$$

where $\Gamma(j)$ is the set of check nodes adjacent to variable node and L_i is the channel LLR.

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \Leftrightarrow$$

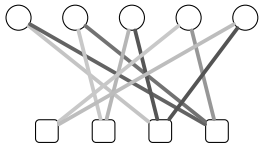

Fig. 1. Parity check matrix and corresponding Tanner graph. Squares – check nodes, circles – variable nodes.

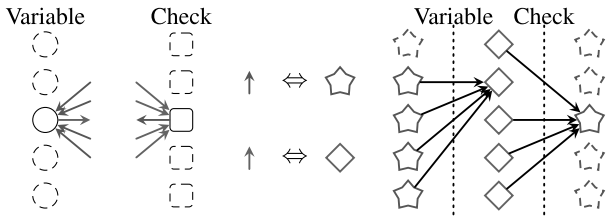

Fig. 2. Tanner graph (left) and corresponding NN (right). NN nodes correspond to messages in a Tanner graph. Two NN vertices are connected if there is a common node in the Tanner graph for corresponding messages.

B. Tanner-based NN

Following [5], we use the NN with the interlayer connections derived from the Tanner graph. There is a one-to-one mapping between the Tanner graph and the NN (also presented as a graph, or NN-graph): each Tanner graph edge corresponds to the NN-graph vertex and vice versa (see Fig. 2). To decode the linear code, one passes messages through the Tanner graph. Thus, each vertex in the NN-graph corresponds to a single message traversed over the original Tanner graph. The NN-graph edges correspond to input and output data required to update the messages (see (3) and (1)). Fig. 2 shows just a single decoding iteration represented by a couple of NN layers: corresponding variable and check layers.

One applies the trainable weights to every message passed through the original Tanner graph [5]. The use of the trainable weights in the NN assumes element-wise operations on the NN-graph vertices. As a result, there are two main differences between conventional NNs with dense layers and Tanner-based NN. First, the connections between layers are sparse, and their structure corresponds to the Tanner graph. Second, the trainable weights are applied differently.

The use of trainable weights $w_{j\to i} > 0$, $\beta_{j\to i} > 0$, $i =$

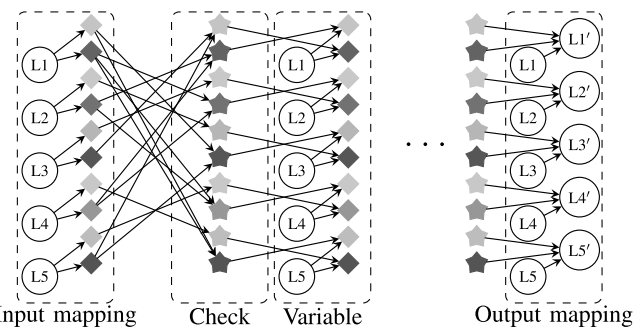

Fig. 3. Sparse NN based on the Tanner graph presented in Fig. 1

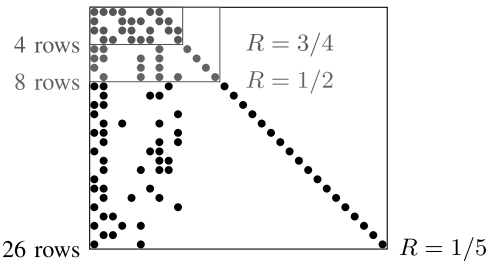

Fig. 4. Base matrix for $K_b = 6$. Markers correspond to nonzero positions. The coding rate sequence consists of $4, \ldots, 26$ rows of base matrix corresponding to coding rates from 0.75 to 0.2.

$1, \ldots, N$, $j = 1, \ldots, M$ results in the following RHS of (2):

$$\prod_{i'} \operatorname{sign}\left(Q_{i' \to j}\right) w_{j \to i} \max\left[\min_{i'}\left\{\left|Q_{i' \to j}\right|\right\} - \beta_{j \to i}, 0\right],$$

where $i' :\in \Gamma(j), i' \neq i$. The variable node rules (3) also become trainable $w^{\text{ch}}_{i \to j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$:

$$Q_{i \to j} = w^{\text{ch}}_{i \to j} L_i + \sum_{j' \in \Gamma(i), j' \neq j} R_{j' \to i}.$$

Given $w^{\text{ch}}_{i \to j} = w_{j \to i} = 1$ and $\beta_{j \to i} = 0$ we have ordinary Min-Sum algorithm, see (2) and (3).

The input mapping presented in Fig. 3 copies the channel LLR multiple times to perform Tanner graph unwrap procedure Fig. 2. The output mapping performs just like a simple variable layer without trainable parameters. Output layer generates the output LLR vector marked by $L1', \ldots L5'$.

C. The weight sharing procedure for 5G codes

The parity check matrix of the 5G LDPC code H consists of square blocks. Each block corresponds to the element in the base parity check matrix H_b, see Fig. 4. Zero elements of the base parity check matrix corresponding to zero blocks of the 5G LDPC parity check matrix, non-zero elements corresponding to the shifted identity matrices called the circulants. Thus, different circulant sizes correspond to different code lengths.

Keeping in mind this structure of 5G LDPC codes, we have reduced the total number of trainable parameters in comparison with original Tanner-based NN. As soon as the party check set remains the same within a single circulant, one can use the single weight for every circulant. This simplification reduces the overall number of trainable parameters by the circulant size, and the training procedure converges faster.

We note that the authors of [14] have introduced the term weight sharing. But, in their case, the weight sharing assumes the same weight value between different iterations rather than within a single circulant.

D. Residual connections

We use residual connections [20] to shortcut to the variable layer outputs. Let the $Q^l_{i \to j}$ be the variable layer output vector at decoder iteration l. The variable layer output is a

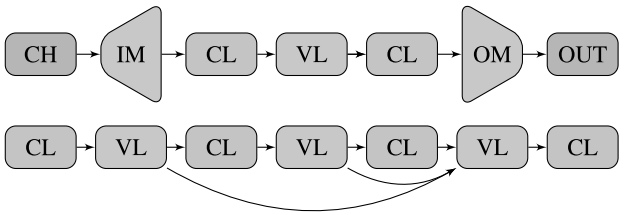

Fig. 5. NN architecture with two decoding iterations (VL – variable layer, CL – check layer), input (IM) and output (OM) mapping (top), residual connections example (see (4)) with the shortcuts taken up to $L = 2$ layers back (bottom).

linear combination of the variable layer outputs from multiple previous layers:

$$Q^l_{i \to j} = w^{\text{ch}}_{i \to j} L_i + \sum_{j' \in \Gamma(i), j' \neq j} R_{j' \to i} + \sum_{t=1}^{D} w^{(l-t)}_{i \to j} Q^{l-t}_{i \to j}, \quad (4)$$

where $(i+1) \mod L = 0, \quad i = 1, \ldots, D$, and L to be the depth of shortcut connections and w^k_{res} are trainable weights for residual connections. Every $(L+1)$-th layer is the residual connections arrival point. This technique helps to confront the vanishing gradient problem [21, 20] and allows us to speed up the training procedure and improve the performance. The NN architecture and residual connections are presented in Fig. 5.

E. Output mapping

The purpose of the output mapping is to construct the output LLRs from the messages traversed over the NN. This procedure is a variable layer without training parameters. Let L_o be the output LLR vector after several decoding iterations (or twice more NN layers). The cross-entropy applied to each element of the $\sigma\left(L_o\right)$ defines the output function, where the $\sigma(\cdot)$ is a sigmoid function that maps LLR value to soft bit value. We use the $\max(\cdot)$ function over the vector of cross-entropy values over the output LLR vector, because our main goal is to minimize the frame error rate (FER).

F. The training procedure

We trained this NN using stochastic gradient descent. As mentioned above, the zero-codeword training property is crucial in the case of an NN-based decoder. Every batch contains the same zero-codeword corrupted by different realizations of additive white Gaussian noise. The noise intensity (or signal-to-noise ratio – SNR) varied within a single batch. The noise intensity was selected at random within the interval at which FER ranges within $10^{-1} \ldots 10^{-4}$.

III. Numerical results

We consider a standard Gray-coded quadrature amplitude shift keying (QPSK) modulation. Every two bits are mapped into a single symbol. The soft demodulation procedure provides the log-likelihood ratio (LLR) vector – the NN input. Consider the LDPC code with $k = 120$ information bits and the coding rate 0.2. The NN performance with weight sharing and residual connections presented in Fig. 6. The scaled

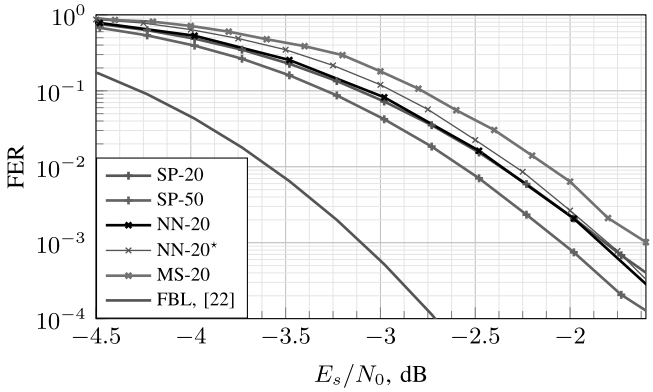

Fig. 6. Decoder performance for LDPC code of length $N = 600$ bits and code rate $1/5$ ($K_b = 6$, $k = 120$). NN-20 is the NN with 20 decoding iterations. Sum-Product (SP) and scaled Min-Sum (MS) with 20 and 50 decoding iterations shown as a reference as well as the finite blocklength (FBL) achievability bound [22].

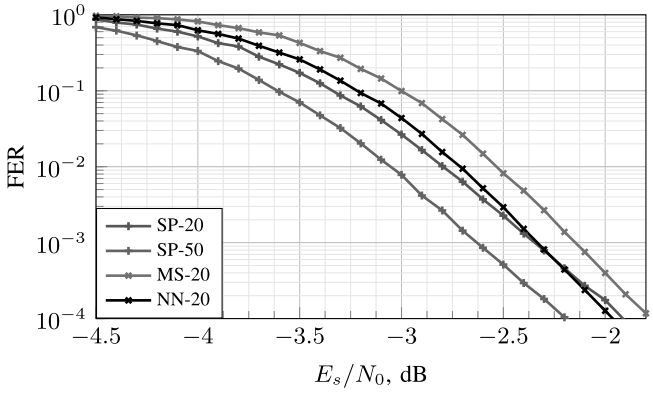

Fig. 7. Decoder performance for LDPC code of length $N = 1200$ bits and code rate $1/5$ ($K_b = 6$, $k = 240$). The weights are taken from the NN trained for $k = 120$ information bits. SP – Sum-product and MS – Min-Sum algorithms performance is provided as a reference.

Min-Sum[1] and Sum-Product performance also presented as a reference. The main achievement for this case is that the Sum-Product algorithm performance achieved with more computationally efficient and tractable Min-Sum (with trainable weights). The finite blocklength achievability bound [22] also presented as a reference. One can observe that this LDPC code is far from the theoretical bounds. The NN was trained with residual connections up to $L = 7$ variable layers back, see Fig. 5. The NN performance trained without residual connections is marked by NN-20*.

IV. Length adaptation

The length adaptation is an approach to generate multiple code lengths while keeping the coding rate constant. The main idea is to keep the base parity check matrix unchanged, and the only variable parameter that adjusts the code length is the circulant size. The weight sharing procedure uses a single weight per circulant. As a result, the optimal weight set remains the same for different circulant sizes. In this section,

[1]The variable node messages are scaled by a single parameter

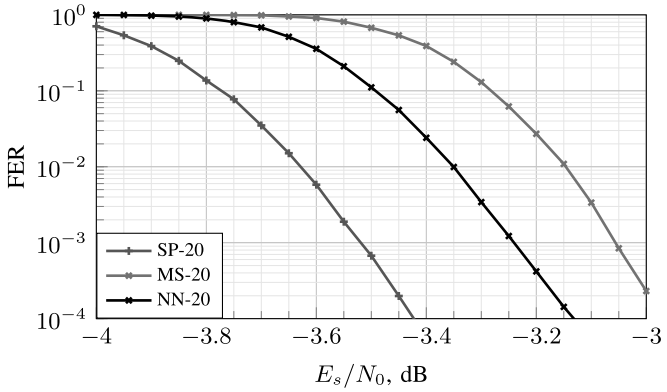

Fig. 8. FER performance for code with $K_b = 10$ and $k = 3840$ with the weights taken from the NN trained for $K_b = 10$ and $k = 200$. A deep NN with 20 iterations marked as NN-20. Sum-Product (SP) and scaled Min-Sum (MS) with 20 decoding iterations performance is provided as a reference.

we validate the length adaptation as follows. We obtain the trained weight on the moderate length. Then, by increasing the circulant size, we evaluate higher code length. We also received the optimal weights for higher length code by training NN without length adaptation and compared the decoding performance of both NNs. Finally, we evaluated the length adaptation for much longer code.

A. Length adaptation for short codes

The NN performance corresponding to code of length 240 bits is presented in Fig. 7. The NN (with the weights taken from $k = 120$ bits NN decoder) outperforms the scaled min-sum algorithm, and further NN training does not improve the decoding performance. As a result, we conclude that the weight set remains optimal for different lifting size values, and one can reuse the same set of weights when changing the lifting size value. We note that the finetuning in the case of $k = 240$ bits is possible on GPU, but longer codes are tricky to train.

B. Length adaptation for moderate length codes

For higher code lengths, we have to disable the residual connections. Residual connections are useful for short code lengths and resulted in faster NN training. But we observed significant performance degradation when the code length increased. As the length of the code increases, the convergence of the decoder slows down. As a result, it is not wise to add LLRs from the previous variable layers.

We have evaluated the decoding performance of the $k = 3840$ and $R = 1/5$ code. (see Fig. 8). We have taken the maximum possible $K_b = 10$ (the number of information bits for the base graph) and extended by the lifting size 384, which gives the code length equal to $N = 19200$. One can observe the performance gain (w.r.t. scaled MS) remaining practically the same even without residual connections.

To deal with the vanishing gradient without residual connections, we have changed the loss function as follows. As soon as every couple of layers correspond to a single decoding iteration, the loss function L_i can be taken after every decoding

TABLE I
PERFORMANCE LOSS TABLE (DB). RED COLOR MARKS CASES PERFORMING WORSE THAN SCALED MIN-SUM

H_b rows count	Weights taken from											
	26	24	22	20	18	16	14	12	10	8	6	4
26	0.00											
24	0.01	0.00										
22	0.03	0.01	0.00									
20	0.03	0.02	0.00	0.00								
18	0.05	0.04	0.02	0.00	0.00							
16	0.08	0.06	0.04	0.03	0.01	0.00						
14	0.08	0.06	0.04	0.03	0.02	0.01	0.00					
12	0.12	0.10	0.07	0.06	0.05	0.04	0.03	0.00				
10	**0.13**	0.11	0.08	0.07	0.06	0.04	0.03	0.02	0.00			
8	**0.21**	**0.18**	**0.15**	**0.13**	0.12	0.10	0.08	0.07	0.06	0.00		
6	**0.29**	**0.27**	**0.23**	**0.20**	**0.19**	**0.14**	**0.12**	0.09	0.07	0.03	0.00	
4	**0.29**	**0.28**	**0.25**	**0.23**	**0.23**	**0.20**	**0.19**	**0.16**	0.15	0.10	0.05	0.00

iteration i (with an additional output layer). The final loss value L is a weighted sum of multiple loss values from different decoding iterations.

$$L = \sum_{i=0}^{D-1} \gamma^i L_i, \quad \gamma \in [0; 1],$$

where D is the nymber of decoding iterations. $i = 0$ corresponds to the last decoding iteration, $i = D-1$ corresponds to the first decoding iteration. We start the training with $\gamma = 1$ to update the weights of deep layers. Then we gradually decrease $\gamma \to 0$. The final NN finetuning performed with a loss function taken just from the last iteration ($\gamma = 0$ given $\gamma^0 = 1$).

We note that the NN performance for short code length slightly degrades when residual connections are disabled, but this allows to reuse the length adaptation for long codes (See Fig. 6, NN-20*).

V. RATE ADAPTATION

The rate adaptation procedure reuses the weights suboptimally to cover multiple coding rates with the same set of trained weights. Fig. 4 provides some insight into how the rate adaptation works. Consider the base graph for the 5G LDPC codes. The higher the coding rate is, the less is the total number of trainable parameters. Moreover, one can construct the weight set corresponding to a higher coding rate as a subset of the lower rate optimal weight set. Our goal is to use the smallest set of optimal weights to maintain all possible coding rates.

Consider some code, optimal weights corresponding to this code, and the SNR value at which frame error rate (FER) matches 10^{-2}. Any suboptimal configuration will deliver 10^{-2} FER at higher SNR. Recall this SNR difference (in dB) as a performance loss.

One can identify the coding rate by the number of rows in the base parity check matrix ($K_b = 6$, see Fig. 4). Table I shows all possible performance loss values. Column index corresponds to the coding rates with optimal weights. Row index corresponds to the coding rate with the weights reused from the code indexed by the column. We constructed this table for the lifting index equal to 20 resulting in $k = 120$ information bits.

For example, the performance loss of code with the rate $1/2$ (corresponds to 8 rows of base parity check matrix) with the weights taken from the optimal weight set corresponding to coding rate $1/5$ (26 rows) is 0.21 dB. Red color marks schemes that underperform the scaled Min-Sum algorithm.

Given different performance loss thresholds, one can define the minimum set of optimal weight required to maintain all coding rates from $3/4$ to $1/5$ ($4, \ldots, 26$ rows of base parity check matrix). For example, given 0.1 dB threshold one needs 3 weight configurations, 0.05 dB requires 6 sets. The full set of weights includes 23 configurations.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have applied the Tanner-based NN decoder to the 5G LDPC decoding problems. We have extended the existing Tanner-based NN with residual connections and weight sharing. Both ideas have reduced the total number of trainable parameters. We found the length and rate adaptation methods to be efficient in the 5G LDPC decoding problem.

As further research, we plan to try different activation functions for the Tanner-based NN as future work. We also plan to apply the deep learning methods to the problem of parity check matrix optimization.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. MIT Press, 1963.

[2] T. Richardson and R. Urbanke, "Multi-edge type LDPC codes," *ISIT talk*, 01 2002.

[3] T. Chen, K. Vakilinia, D. Divsalar, and R. D. Wesel, "Protograph-Based Raptor-Like LDPC Codes," *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1522–1532, May 2015.

[4] 3GPP, "NR; Multiplexing and channel coding," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.212, 09 2017, version 1.0.0.

[5] 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, Feb 2018.

[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*, Oct 1995, pp. 89–.

[7] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *proc. of IEEE Global Communications Conference (GLOBECOM)*, Dec 2017, pp. 1–6.

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

[9] 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)*, March 2017, pp. 1–6.

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

[11] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, January 1972.

[12] M. P. C. Fossorier and Shu Lin, "Soft decision decoding of linear block codes based on ordered statistics for the Rayleigh fading channel with coherent detection," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 12–14, Jan 1997.

[13] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, "RNN Decoding of Linear Block Codes," 2017. [Online]. Available: https://arxiv.org/abs/1702.07560

[14] M. Lian, F. Carpi, C. Häger, and H. D. Pfister, "Learned belief-propagation decoding with simple scaling and snr adaptation," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 161–165.

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

[16] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[17] E. Nachmani and L. Wolf, "A gated hypernet decoder for polar codes," in *proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 5210–5214.

[18] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Learn codes: Inventing low-latency codes via recurrent neural networks," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 207–216, 2020.

[19] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

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

[22] Y. Polyanskiy, H. V. Poor, and S. Verdu, "Channel Coding Rate in the Finite Blocklength Regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, May 2010.