Multi Variable-layer Neural Networks for Decoding Linear Codes

Samira Malek, Saber Salehkaleybar, and Arash Amini
Department of Electrical Engineering, Sharif University of Technology, Tehran , Iran
Email: Maalek.Samira@ee.sharif.edu, {saleh,aamini}@sharif.edu

***Abstract*—The belief propagation algorithm is a state of the art decoding technique for a variety of linear codes such as LDPC codes. The iterative structure of this algorithm is reminiscent of a neural network with multiple layers. Indeed, this similarity has been recently exploited to improve the decoding performance by tuning the weights of the equivalent neural network. In this paper, we introduce a new network architecture by increasing the number of variable-node layers, while keeping the check-node layers unchanged. The changes are applied in a manner that the decoding performance of the network becomes independent of the transmitted codeword; hence, a training stage with only the all-zero codeword shall be sufficient. Simulation results on a number of well-studied linear codes, besides an improvement in the decoding performance, indicate that the new architecture is also simpler than some of the existing decoding networks.**

***Index Terms*—Belief propagation, linear codes, neural networks.**

I. Introduction

In recent years, deep learning has marked significant improvements in a wide range of applications such as speech processing [1], playing games [2], and image recognition [3]. While the success of deep learning is primarily reported in complex classification and reconstruction tasks that lack a well-defined mathematical model, it is being deployed even in settings with studied classical models. One such example is the decoding of linear binary codes. The latter is one of the oldest problems in information theory, and could be considered as a classification problem with exponentially many classes (either the number of codewords or the number of distinguishable error patterns).

By casting the decoding challenge as a deep learning problem, we essentially need a neural network to learn how to optimally decode an arbitrary vector, after being trained. Ideally, the training data shall include the space of all codewords. Nevertheless, due to the exponential increase of the number of codewords in terms of the code length, having access to such a big training dataset is not feasible in moderate to large code lengths. Besides, the neural network that can efficiently decode such codes shall consist of a large number of parameters. Consequently, a naive neural network decoder is only applicable to codes with short lengths [4]–[7].

The belief propagation (BP) algorithm is a generic iterative decoding process that takes into account the Tanner graph of the code. The Tanner graph consists of two categories of nodes: check nodes and variable nodes. The BP algorithm is a message passing technique in which certain messages are iteratively sent along the edges between the check and variable nodes. This algorithm is known to be optimal when the Tanner graph is a tree (which almost never happens) [8], though it has acceptable performance even for non-tree Tanner graphs [9]. This algorithm is considered as the state of the art decoder for binary codes such as LDPC codes that do not have short cycles in their Tanner graphs.

The iterative nature of the BP algorithm is recently interpreted as a neural network structure in [10], [11]. The layers of the neural network are associated with the messages sent by either the check nodes or the variable nodes. Hence, the number of layers is roughly twice the number of BP iterations. The advantage of the latter interpretation is that weights in the neural network can now be tuned to yield the optimal performance, in contrast to the standard BP that uses universally fixed weights and structure for all codes. To avoid the need for huge training datasets, certain restrictions are applied to the structure of the neural network in [10], [11] that makes the training (and the outcome of the neural network) independent of the original codeword. As a result, only noisy versions of the zero codeword are sufficient for the training stage. A similar approach is devised in [12] for the min-sum algorithm and the effect of the quantization (implemented using a neural network) is studied in [13]. An extension of the approach to hyper-graph-networks is introduced in [14] with a dynamically hard training process and considerably more parameters. To meet the computational cost of the latter hypernetwork, a hardware implementation is proposed in [15].

In this paper, we introduce a new neural network structure for decoding linear binary codes that performs superior to [10], [11] without adding computational complexity. More specifically, our structure consists of multiple (mainly two) variable-layers in succession before the check-layers. We further impose the same constraints as in [10] to avoid the need for large training datasets; i.e., the training of the proposed network is accomplished solely via the noisy versions of the zero codeword.

The rest of the paper is organized as follows. In Section II, we describe the BP algorithm and its equivalent neural network. Then, in Section III, we propose our structure and its training details. We validate our method via simulation results in Section IV. The results indicate superiority of the performance of the proposed method compared to the existing methods in the literature. Finally, we conclude the paper in Section V.

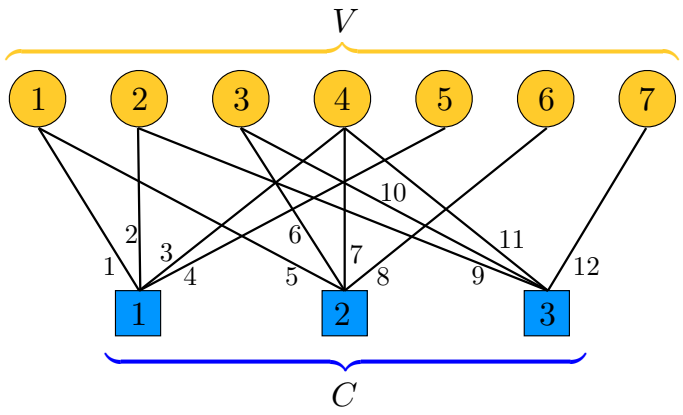

Fig. 1. The Tanner graph of the $(7, 4)$ Hamming code.

II. Background

Before proposing our method, we briefly review the belief propagation algorithm, which is used for decoding linear codes. Then we describe neural network form of this algorithm.

A. Tanner Graph

For a (n, k) linear code, the Tanner graph of the code is a bipartite graph that consists of n variable nodes and $n - k$ check nodes [16]. The i-th check node is connected to the j-th variable node, if the (i, j) element in the parity check matrix of the code is equal to one. As an illustrative example, the Tanner graph of the $(7, 4)$ Hamming code with parity check matrix

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

is shown in Fig. 1.

B. Belief Propagation Decoding

The BP decoding is an iterative technique over the Tanner graph of the code [17]–[19]. In each iteration of the BP algorithm, probability estimates or log-likelihood ratios (messages) are communicated over the graph between variable and check nodes.

By denoting the channel output (noisy codeword) as y and the original codeword as x, the initialization for the log-likelihood ratios (LLR) in the variable nodes is given by

$$\mu_v = \log\left(\frac{pr(x_v = 1|\, y_v)}{pr(x_v = 0|\, y_v)}\right), \tag{1}$$

where v stands for the bit index. Next, the LLRs are iteratively sent back and forth between the variable and check nodes.

The message $\mu_{v,c}^{(l)}$ in the l-th iteration of the BP algorithm that is sent from variable node v to the check node c is given by

$$\mu_{v,c}^{(l)} = \mu_v + \sum_{c' \in C_v \backslash \{c\}} \mu_{c',v}^{(l-1)}, \tag{2}$$

where C_v represents the set comprising of all check nodes incident to the variable node v and $\mu_{c,v}^{(-1)} = 0$ at $l = 0$. Similarly, the message $\mu_{c,v}^{(l)}$ in the second phase of the l-th iteration that is sent from check node c to variable node v is

$$\mu_{c,v}^{(l)} = 2\text{Arctanh}\left(\prod_{v' \in V_c \backslash \{v\}} \tanh\left(\frac{\mu_{v',c}^{(l)}}{2}\right)\right), \tag{3}$$

where V_c is the set of all variable nodes connected to the check node c.

Finally, the estimates of posterior LLRs after the l-th iteration are computed using

$$o_v^{(l)} = \mu_v + \sum_{c' \in C_v} \mu_{c',v}^{(l)}, \tag{4}$$

C. Deep Neural Belief Propagation

Here, we describe the deep network structure of [10], [11] that mimics the BP algorithm. With the weights incorporated in this structure, the network is able to generate the exact output of the BP algorithm (no tuning of the weights), or even generate better results by tuning the weights. While we can still consider the neural network as a message passing approach on the Tanner graph, the effective messages are no longer the same as the weight-less case. Indeed, the messages in (2) and (3) are updated as

$$\mu_{v,c}^{(l)} = \tanh\left(\frac{1}{2}\left(b_{v,c}^{(l)}\mu_v + \sum_{c' \in C_v \backslash \{c\}} \omega_{v,c,c'}^{(l)} \mu_{c',v}^{(l-1)}\right)\right), \tag{5}$$

and

$$\mu_{c,v}^{(l)} = 2\text{Arctanh}\left(\prod_{v' \in V_c \backslash \{v\}} \mu_{v',c}^{(l)}\right), \tag{6}$$

respectively. Note that $b_{v,c}^{(l)}$s and $\omega_{v,c,c'}^{(l)}$s are the parameters (weights) introduced by the neural network; if they are all set to 1, we obtain the classical BP algorithm. In a similar way, the LLR estimates in (4) are updated via

$$o_v = \sigma\left(b_v^{out}\mu_v + \sum_{c' \in C_v} \omega_{c',v}^{out} \mu_{c',v}^{(l)}\right), \tag{7}$$

where $\sigma(\cdot)$ is a sigmoid function converting the LLRs into probabilities.

The number of nodes in each hidden layer of the neural network equals the number of edges in the Tanner graph; for instance, each hidden layer in the neural network of the Hamming code in Fig. 1 has 12 nodes. Based on the message type generated by each layer, it is called variable-layer (referring to equation (5)), check-layer (referring to equation (6)) or marginal-layer (referring to equation (7)). As the nodes in each layer represent edges in the Tanner graph, the connections between consecutive layers are determined based on the adjacency of the edges in the Tanner graph; i.e., two nodes in consecutive layers of variable-layer and check-layer (check-layer and variable-layer) are connected if the corresponding edges in the Tanner graph share a vertex in variable nodes (check nodes). It is not difficult to show that

the message passing symmetry condition of [18] is fulfilled in this network. As a result, the network could be trained based on the noisy versions of only a single codeword.

III. Proposed Method

In this section, we introduce a new neural architecture for decoding that both improves the performance in terms of the bit error rate (BER) and reduces the computational complexity of decoding with respect to the competing methods. This new network is easily trainable and satisfies the message passing symmetry conditions. Hence, it can be trained with only the noisy versions of the zero codeword.

A. Multi variable-layer network (MVN)

We propose to replace the variable-layers with multiple (oftentimes 2) cascaded copies of a variable-layer. We call the proposed neural decoder "multi variable-layer network (MVN)". For instance, by cascading two variable-layers in l-th iteration, equations (2), (3), and (4) are replaced with the following equations

For $l = 0$:

$$\mu_{v,c}^{(0)} = \mu_v, \tag{8}$$

$$\mu_{c,v}^{(0)} = 2\text{Arctanh}\left(\prod_{v' \in V_c \setminus \{v\}} \tanh\left(\frac{\mu_{v',c}^{(0)}}{2}\right)\right), \tag{9}$$

$$o_v^{(0)} = \mu_v + \sum_{c' \in C_v} \omega_{c',v}^{(0)} \mu_{c',v}^{(0)}, \tag{10}$$

and for $l \geq 1$:

$$\mu_{v,\tilde{v},c}^{(l)} = \tanh\left(\frac{1}{2}\left(b_{v,c}^{(l)}\mu_v + \sum_{c' \in C_v \setminus \{c\}} \omega_{v,c,c'}^{(l)} \mu_{c',v,\tilde{v}}^{(l-1)}\right)\right), \tag{11}$$

$$\mu_{\tilde{v},c,v}^{(l)} = \tanh\left(\frac{1}{2}\left(\tilde{b}_{v,c}^{(l)}\mu_v + \sum_{c' \in C_v \setminus \{c\}} \tilde{\omega}_{v,c,c'}^{(l)} \mu_{v,\tilde{v},c'}^{(l)}\right)\right), \tag{12}$$

$$\mu_{c,v,\tilde{v}}^{(l)} = 2\text{Arctanh}\left(\prod_{v' \in V_c \setminus \{v\}} \mu_{\tilde{v},c,v'}^{(l)}\right), \tag{13}$$

$$o_v^{(l)} = \mu_v + \sum_{c' \in C_v} \omega_{c',v}^{(l)} \mu_{c',v,\tilde{v}}^{(l)}, \tag{14}$$

where $\mu_{c',v,\tilde{v}}^{(0)} = \mu_{c,v}^{(0)}$, and the variables in the above equations are depicted in Fig. 2(a).

As described in Section II, the BP algorithm is an iterative technique that each iteration consists of two parts. In the first part, messages computed in variable nodes are sent to check nodes and in the second part, updated messages in check nodes are sent back to variable nodes. In our proposed MVN, the first part is modified as the number of variable-layers increases. In particular, except for the first iteration, we apply two variable-layers both consisting of weights. While the first variable-layer receives its messages from a check-layer (and acts as before), the second variable-layer acts on the received messages from the first variable-layer. However, in terms of the action, the second layer pretends that the messages are received from a check-layer and are to be sent to the next check-layer.

We should highlight that the number of nodes in each hidden layer of MVN equals the number of edges in the Tanner graph. The connections among the nodes of neighboring layers, are determined based on the adjacency of the corresponding edges in the Tanner graph. In particular, the node i in the l-th layer is connected to the node j in the $l+1$-th layer if the edges i and j in the Tanner graph share a vertex of the type of the $l+1$-th layer; more precisely, if the $l+1$-th layer is a variable-layer, the shared vertex shall be a variable node, and if the $l+1$-th layer is a check-layer, the shared vertex shall be a check node.

It can be shown that this new architecture preserves the message passing symmetry condition in [18] since we repeat a process that itself fulfills the symmetry condition [10]. Thus, its performance is independent of the transmitted codewords and it can be trained with noisy versions of the zero codeword.

It should be noted that it is possible to use different numbers of variable-layers in each iteration. For example, we could have a network with three iterations which is consist of one variable-layer in the first iteration, three variable-layers in the second iteration, and two variable-layers in the third iteration.

We consider the cross entropy multi-loss function [11], [14] for training the proposed neural network:

$$\Gamma = -\frac{1}{N}\sum_{i=a}^{L}\sum_{v=1}^{N} x_v \log\left(o_v^{(i)}\right) + (1 - x_v)\log\left(1 - o_v^{(i)}\right) \tag{15}$$

where $x_v = 0$, $a \in \{0, 1\}$, and $o_v^{(i)}$ is the output of i-th iteration, defined in (14) and (10).

IV. Experiments

We used Tensorflow environment in order to implement our method and applied it on BCH(127,99), BCH(127,64), BCH(63,36) and BCH(63,45) codes. First, we present simulations results for codes with length 127 and then for the codes with length 63.

A. Results for BCH code with N = 127

The training dataset was generated by sending zero codeword through additive white Gaussian noise (AWGN) channel. The signal-to-noise-ratio (SNR) of the channel was varied from 1 dB to 8 dB. Each mini-batch included 40 noisy code samples constructed from 5 codewords per SNR. Moreover, parity check matrices based on [20] were used to form the connections between nodes in the network. For training the network, the RMS-Prop optimizer [21] with learning rate of 0.003 was utilized. Our designed networks for BCH(127,99) and BCH(127,64) consist of three iterations with two consecutive variable-layers. For BCH(127,99), the parameter a in (15) was set to 0 and $\omega_{c',v}^{(l)}$s in (14) were not trained. For BCH(127,64), the parameter a in (15) was set to 0 and $\omega_{c',v}^{(l)}$s in (14) only for the last iteration, i.e., $l = 3$, were trained.

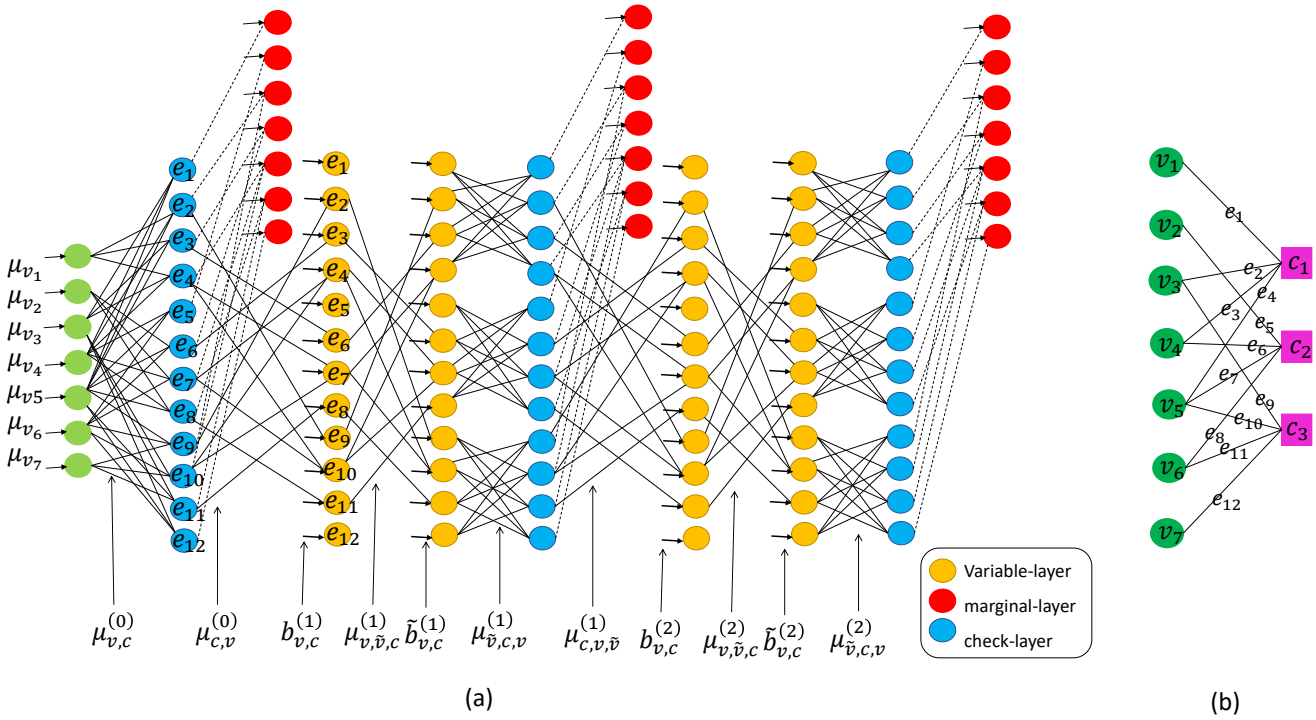

Fig. 2. (a) A diagram of MVN architecture for BCH(7,4) with 3 iterations. The last two iterations have two variable-layers. (b) The Tanner graph of BCH(7,4) code.

Figures 4 and 5 illustrate results for BCH(127,99) and BCH(127,64). By using three double variable-layers, we can eliminate two complex check-layers that include many multiplications. Comparing to [11], we reduce the BER by factors up to 3.54 and 6.3, for BCH(127,99) and BCH(127,64), respectively.

Figures 3 (a), (b) depict the total number of trained weights and multiplications that are used for decoding one codeword with our method and the proposed networks in [11]. Multiplications are costly in hardware implementation and it is important to reduce the number of multiplications. Figures 3 (a), (b) show that the number of trained weights and the number of multiplications are reduced with respect to the ones in [11]. It is noteworthy to mention that two consecutive variable-layers' weights and biases were trained for BCH(127,99) code and two consecutive variable-layers' weights and biases, and final marginal-layer's weights were trained for BCH(127,64) code.

B. Results for BCH with N = 63

The training dataset was again generated from noisy versions of the zero codeword with the following values of SNR: $\{1, 2, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 8\}$ dB. Each mini-batch included 120 noisy code samples constructed from 10 codewords per SNR. Moreover, parity check matrices based on [20] were used. For BCH(63,45), hidden layers consisted of five iterations in which the first four iterations had double variable-layers and the last one had a single variable-layer. For the BCH(63,36) code, hidden layers consisted of five iteration in which the first three iterations had double variable-layers and the last two iterations had a single variable-layer. For both BCH(63,45) and BCH(63,36), the parameter a in (15) was set to 1. For the training of these networks, RMS-Prop optimizer was utilized with learning rate of 0.001.

Figures 6 and 7 present the results for BCH(63,45) and BCH(63,36), respectively. As it is observed, the BER is reduced by a factor up to 5.25 and 2.12 compared to [11], for BCH(63,45) and BCH(63,36), respectively. Moreover, according to Fig. 3 (c), (d), the number of weights and multiplications in our network are slightly higher than the ones in [11] and significantly less than the ones of [14]. In Table I, the negative natural logarithm of BER is reported for SNR=$4, 5, 6$ dB for our proposed network and the two Hyper-networks (small f and large f) proposed in [14]. For BCH(63,36), our proposed network performs better than the Hyper-network (small f) [14] and Hyper-network (large f) [14]. Furthermore, for BCH(63,45), the performance of our proposed network is better than Hyper-network (large f) [14] and worse than Hyper-network (small f) [14].

Overall, the number of weights in the proposed networks for BCH(63,45) and BCH(63,36) is slightly higher than the networks proposed in [11], but they improve the BER with respect to [11]. Although Hyper-networks [14] improve the performance of [11], they have dynamically hard training process as mentioned in [14] while MVN is easily trainable due to the simplicity of the structure. Furthermore, the BER of our proposed network is less than the ones of Hyper-networks for BCH(63,36).

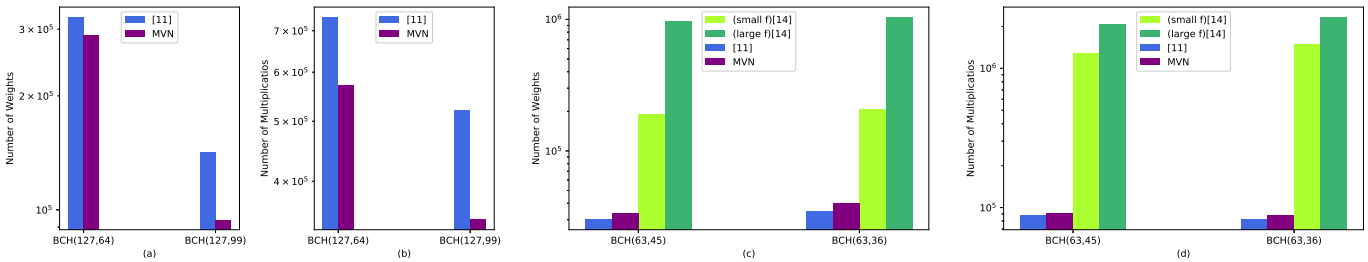

Fig. 3. The number of trained weights and multiplications between [11] and double-variable network for (a) BCH(127,99) and (b) BCH(127,64). The number of trained weights and multiplications between [11], [14], and double-variable network for (c) BCH(63,36) and (d) BCH(63,45).

TABLE I
THE NEGATIVE NATURAL LOGARITHM OF BIT ERROR RATE (BER) FOR THREE SNRS $4, 5, 6$ DB OF OUR METHOD AND PROPOSED NETWORKS IN [14]. HIGHER IS BETTER.

method	(small f) [14]			(large f) [14]			MVN		
SNR	4	5	6	4	5	6	4	5	6
BCH(63,36)	3.96	5.35	7.20	4.00	5.42	7.34	4.05	5.45	7.55
BCH(63,45)	4.48	6.07	8.45	4.41	5.91	7.91	4.49	5.96	8.08

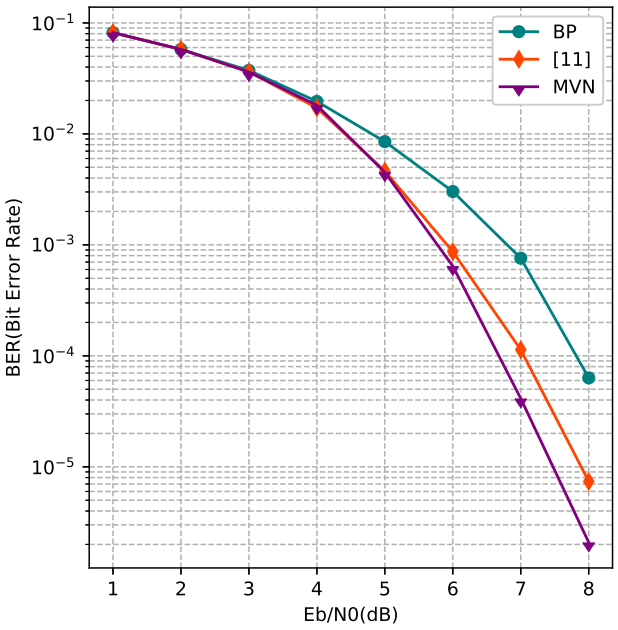

Fig. 4. BER results for BCH(127,99) code.

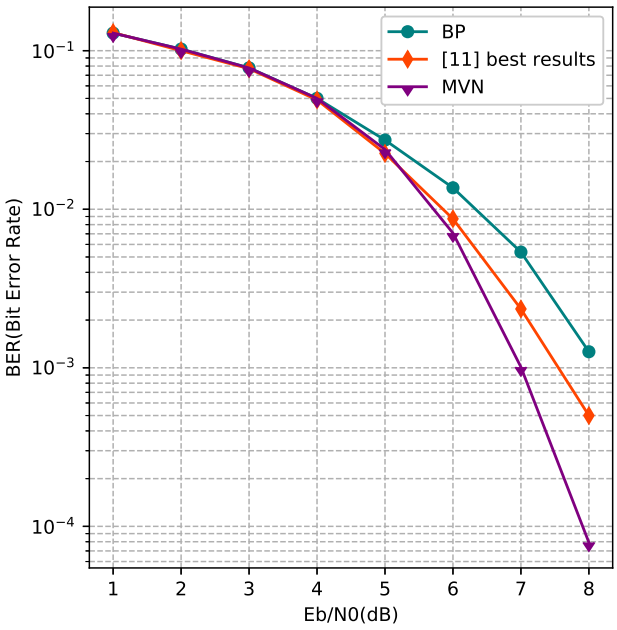

Fig. 5. BER results for BCH(127,64) code.

V. CONCLUSION

In this paper, we proposed a new architecture for the neural decoders. The proposed architecture preserves the message passing symmetry condition. Hence, its performance is independent of the transmitted codewords and it could be trained with merely noisy zero codewords. In this architecture, we added multiple variable-layers in each iteration which results in eliminating some complex check-layers. For instance, for BCH codes with N = 127, these added layers generally reduce the number of trained weights since fewer iterations are needed to get the same performance. Moreover, experimental results show that multi variable-layer network performs better than existing methods in terms of BER.

REFERENCES

[1] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams *et al.*, "Recent advances in deep learning for

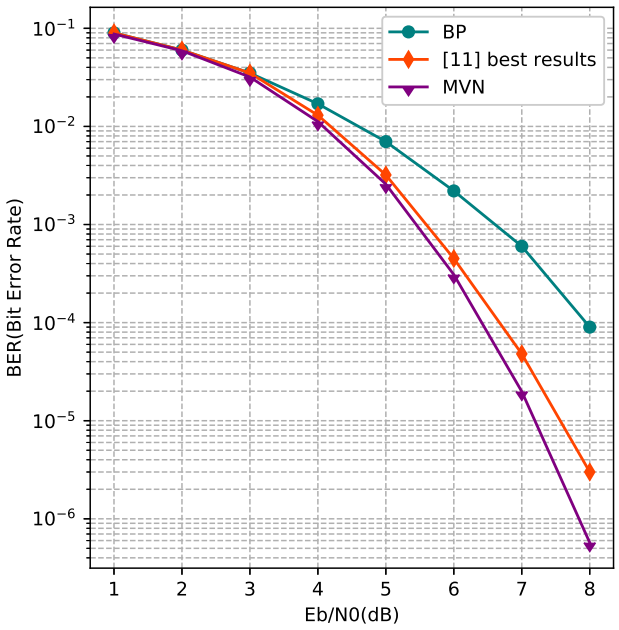

Fig. 6. BER results for BCH(63,45) code.

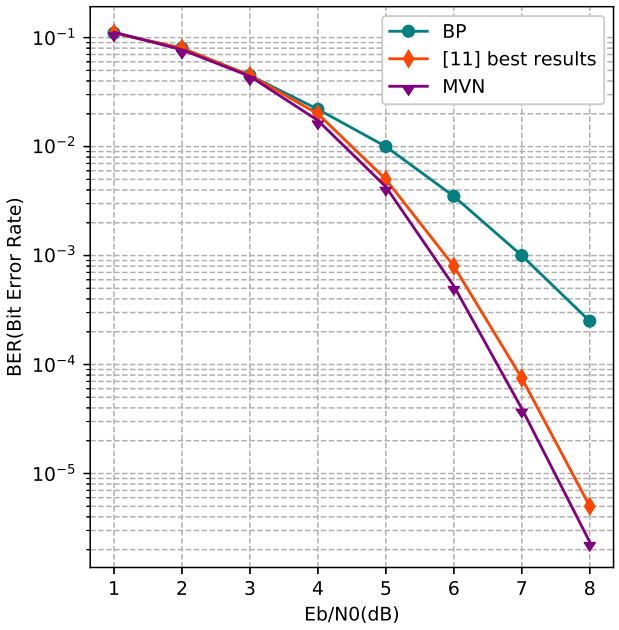

Fig. 7. BER results for BCH(63,36) code.

speech research at microsoft," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8604–8608.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] J. Bruck and M. Blaum, "Neural networks, error-correcting codes, and polynomials over the binary n-cube," *IEEE Transactions on information theory*, vol. 35, no. 5, pp. 976–987, 1989.

[5] Y.-H. Tseng and J.-L. Wu, "High-order perceptrons for decoding error-correcting codes," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 3. IEEE, 1992, pp. 24–29.

[6] J.-L. Wu, Y.-H. Tseng, and Y.-M. Huang, "Neural network decoders for linear block codes," *International Journal of Computational Engineering Science*, vol. 3, no. 03, pp. 235–255, 2002.

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

[8] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-82)*, 1982, pp. 133–136.

[9] ——, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[10] 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)*. IEEE, 2016, pp. 341–346.

[11] 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.

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

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

[14] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," in *Advances in Neural Information Processing Systems*, 2019, pp. 2326–2336.

[15] M. Rotman and L. Wolf, "Electric analog circuit design with hypernetworks and a differential simulator," *arXiv preprint arXiv:1911.03053*, 2019.

[16] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[17] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[18] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[19] A. Shokrollahi, "Ldpc codes: An introduction," *Digital Fountain, Inc., Tech. Rep*, vol. 2, p. 17, 2003.

[20] 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.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.