A Neural Network Lattice Decoding Algorithm

Mohammad-Reza Sadeghi, Farzane Amirzade, Daniel Panario, and Amin Sakzad

arXiv:1807.02913v2 [cs.IT] 13 Sep 2018

Abstract-Neural network decoding algorithms are recently introduced by Nachmani et al. to decode high-density paritycheck (HDPC) codes. In contrast with iterative decoding algorithms such as sum-product or min-sum algorithms in which the weight of each edge is set to 1, in the neural network decoding algorithms, the weight of every edge depends on its impact in the transmitted codeword. In this paper, we provide a novel feed-forward neural network lattice decoding algorithm suitable to decode lattices constructed based on Construction A, whose underlying codes have HDPC matrices. We first establish the concept of feed-forward neural network for HDPC codes and improve their decoding algorithms compared to Nachmani et al. We then apply our proposed decoder for a Construction A lattice with HDPC underlying code, for which the well-known iterative decoding algorithms show poor performances. The main advantage of our proposed algorithm is that instead of assigning and training weights for all edges, which turns out to be timeconsuming especially for high-density parity-check matrices, we concentrate on edges which are present in most of 4-cycles and removing them gives a girth-6 Tanner graph. This approach, by slight modifications using updated LLRs instead of initial ones, simultaneously accelerates the training process and improves the error performance of our proposed decoding algorithm.

Index Terms—Lattices, deep learning, Tanner graph, trellis graph.

I. INTRODUCTION

C Onstructing lattices from codes has been an intense research topic which resulted in well-know Constructions A, B, C, D, and D' (see [1] and [2]). High dimensional lattices along with iterative decoding algorithms were first introduced by Sadeghi et al. [3]. Although high dimensional lattices have high coding gain, in practice, they are difficult to implement and in higher dimensions they have high decoding complexity. Several decoding algorithms have been proposed in the literature. The generalized min-sum algorithm was presented in [3] to decode low-density parity-check (LDPC) lattices. Other iterative decoding algorithms to decode LDPC lattices are sum-product algorithm (SPA) [4] and FFT based SPA [5]. In these methods, the process of transmitting messages in variable node and check node operations is applied on the Tanner graph of LDPC lattices.

Some underlying codes of the algebraic lattices have a rather high-density parity-check (HDPC) matrix. For example, the underlying code of Barnes-Wall lattices are Reed-Muller codes whose parity-check matrices are not sparse. These lattices are based on Construction D' [6] using a set of nested Reed-Muller codes. In this case, the variable-node and check-node operations in message passing decoding algorithms are timeconsuming. In fact, applying iterative algorithms to decode most well-known algebraic codes would result in poor performance when compared to maximum likelihood decoders [7]. To resolve this problem, deep neural network decoders were proposed [8], [9].

In recent years, deep learning methods have shown amazing performances in a variety of subjects. For example, an application of deep learning methods to the problem of low complexity channel decoding has been proposed [9]. It has been demonstrated that deep learning methods improve the min-sum and sum-product algorithms for HDPC codes using a weighted trellis graph, which is another representation of codes. The main benefit of the neural network decoder is that the weight of every edge relies on its influence in the transmitting messages. By setting weights properly, we can compensate for small cycles, which are the main causes of high error floor regions and deterioration of the decoding process. The first step in this method is to train weights for a given codeword like the all-zero codeword 0. Then, using these trained weights, we decode codewords. The computation complexity of the neural network decoder depends on the number of weights required to be trained. For example in [8], this number is 2n + 2LE, where n is the number of variable nodes, L is the number of iterations and E is the number of edges in the corresponding trellis diagram of the code.

In the ℓ -th iteration of trellis graph in [8], there are two hidden layers VN_{ℓ} and CN_{ℓ} for the variable node operation and the check node operation, respectively. In this framework, two weights w_ℓ and w'_ℓ are defined for each edge in the ℓ -th iteration. The weight w_{ℓ} links the first layer and the hidden layer, CN_{ℓ} , and the weight w'_{ℓ} links the hidden layers $CN_{\ell}, VN_{(\ell+1)}$. In [9], a new recurrent neural network (RNN) algorithm was defined by which the number of weights to train was significantly reduced to 2E. In this method, all weights between the first layer and hidden layers CN_{ℓ} , for $1 \leq \ell \leq L$, are set to 1. No training process is needed for them as the experimental results have shown that training these parameters did not result in any performance improvement. Moreover, for each edge in the RNN method, two weights w and w' are considered fixed in iterations. In this framework, w is the weight associated to the hidden layers CN_{ℓ} , $VN_{(\ell+1)}$ and w' is the weight associated to the hidden layer CN_{ℓ} and the layer related to hard decision operation of the ℓ -th iteration. Numerical results [9] show that the RNN method outperforms the neural network decoding algorithm in [8].

M.-R. Sadeghi and F. Amirzade are with the Faculty of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran. D. Panario is with the School of Mathematics and Statistics, Carleton University, Ottawa, Canada. A. Sakzad is with the Clayton School of IT, Monash University, Melbourne, Australia. (e-mails: msadeghi@aut.ac.ir, famirzade@gmail.com, daniel@math.carleton.ca, amin.sakzad@monash.edu).

In a concurrent work, a deep learning lattice decoder is also proposed in [12]. In this paper, we aim to present a neural network decoding algorithm for lattices constructed based on Construction A, whose underlying codes have highdensity parity-check matrices. All the weights w' in the RNN algorithm are trained in this work. However, instead of considering the weight w for each edge, similar to the RNN algorithm, we mainly focus on edges which are, potentially, the main culprits of high decoding failure rates. In particular, we define weights w only for the edges that are present in most of the 4-cycles in the Tanner graph of the code and by removing them the girth of the Tanner graph is increased to 6. We call such edges "culprit edges". The weight for the non-culprit edges are considered to be 1. So, the pair of weights to train for culprit edges is (w, w') and for the other edges is set to be (1, w'). By properly training these weights and running SPA with the trained weights, we achieve an improvement in the decoding process of lattices constructed based on Construction A, whose underlying code is a HDPC code.

The rest of the paper is organized as follows. In Section II, we give some basic notations and definitions. In Section III, we present three contributions: first, the structure of the trellis graph, which is completely new and different from the ones in the literature; second, the message computations in variable node and check node operations on the trellis graph; and third, a few algorithms to train weights in the neural network decoding algorithm. In Section IV, we apply our newly proposed deep neural network decoding algorithm to a lattice based on Construction A. We finish our paper in the last section giving concluding remarks and open problems.

II. PRELIMINARIES

A lattice, Λ , is a discrete additive subgroup of \mathbb{R}^m [2]. The set of linearly independent vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$, where nis the dimension of the lattice, is a lattice basis. The generator matrix of the lattice is defined as $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n]^T$ and a lattice can be shown by $\Lambda = \{\mathbf{v} = \mathbf{xB} : \mathbf{x} \in \mathbb{Z}^n\}$. We let m = n and consider only full dimensional lattices in this paper. The length of the shortest nonzero vector of the lattice is denoted by d_{\min} with respect to Euclidean norm. The notation det (Λ) is used to denote the volume of the lattice which is obtained by det (\mathbf{BB}^T) , and V_n is the volume of the *n*dimensional sphere of radius 1. For an unconstrained additive with Gaussian noise (AWGN) channel, the volume-to-noise ratio of the lattice Λ is defined by VNR = $\frac{(\det(\Lambda))^{\frac{2}{n}}}{(\det(\Lambda))^{\frac{2}{n}}}$.

ratio of the lattice Λ is defined by $\text{VNR} = \frac{(\det(\Lambda))^{\frac{2}{n}}}{2\pi e \sigma^2}$. Suppose $C \subseteq \mathbb{F}_p^n$ is a linear code over \mathbb{F}_p^n , where p is a prime number. We construct Λ as follows: $\mathbf{x} = (x_1, \ldots, x_n)$ is a point of lattice if and only if there exists a codeword $\mathbf{c} = (c_1, \ldots, c_n)$ such that $\mathbf{x} \equiv \mathbf{c} \mod p$. In other words,

$$\Lambda = p\mathbb{Z}^n + C = \{ (pz_1 + c_1, \dots, pz_n + c_n) \colon z_i \in \mathbb{Z}, \mathbf{c} \in C \},$$
(1)

where the components of c in (1) are considered as real numbers and operations are done in reals. The produced lattice Λ is said to be constructed based on Construction A [2].

III. TRELLIS GRAPH AND BACK-PROPAGATION ALGORITHM

Every linear code with parity check matrix **H** can be represented by a Tanner graph which is a bipartite graph whose incident matrix is **H**, see Example 1. In the existing iterative decoding algorithms in the literature, a message passing process is considered on the Tanner graph. The SPA is a message passing algorithm that exchanges messages through the edges in the corresponding Tanner graph. In this section, we first explain SPA on Tanner graphs and trellis diagrams and then we introduce our back-propagation algorithm.

A. Sum-Product Algorithm on Tanner Graph

Let us first recall a SPA on Tanner graph of a code. Each iteration in SPA consists of two main processes, variable node process and check node process. Let v_i , ch_j , c_v and y_v denote the *i*-th variable node, the *j*-th check node, the *v*-th component of the transmitted codeword and the *v*-th channel output, respectively. Nodes v_i , $1 \le i \le n$, are associated with the initial inputs which are the log-likelihood ratios (LLR) of the channel output given by

$$\ell_v = \log \frac{\Pr(c_v = 1|y_v)}{\Pr(c_v = 0|y_v)}.$$
(2)

In the variable node operation, the message μ_{v_i,ch_j} from *i*-th variable node to *j*-th check node is computed by

$$\mu_{v_i,ch_j} = \ell_v + \sum_{ch_k \in V_i, \ k \neq j} \mu_{ch_k,v_i},\tag{3}$$

where V_i is the set of all neighboring check nodes to the variable node v_i . In the check node operation, the message μ_{ch_i,v_i} from the *j*-th check node to the v_i is computed by:

$$\mu_{ch_j,v_i} = 2 \tanh^{-1} \left(\prod_{v_k \in CH_j, \ k \neq i} \tanh\left(\frac{\mu_{v_k,ch_j}}{2}\right) \right), \quad (4)$$

where CH_j denotes the set of all neighboring variable nodes to the *j*-th check node ch_j . After each check node operation we have a hard decision process in which we obtain updated LLRs, denoted by o_v , for $1 \le v \le n$, and computed as follows:

$$o_v = \ell_v + \sum_{ch_k \in V_i} \mu_{ch_k, v_i}.$$
(5)

If $o_v < 0$, then $c_v = 0$, and if $o_v > 0$, then $c_v = 1$. If $\mathbf{H}\mathbf{c}^T = 0$, then the decoding process halts and returns the decoded codeword.

B. Sum-Product Algorithm on Trellis Graph

In the neural network decoding algorithms proposed in [8] and [9], the message passing process is taken place on the trellis graph corresponding to the Tanner graph. The number of hidden layers with L full iterations is 2L; with the input and output layers there are a total of 2L + 2 layers in the trellis graph. In the following, we introduce a new trellis graph representation, which is different from the ones in the literature.



Fig. 1. The Tanner graph of H in Example 1.

Let n and E be the number of variable nodes and the number of edges in the Tanner graph of a linear code, respectively. If L is the number of iterations, then the trellis graph has 3L + 1 layers. The first layer consists of n nodes. There are three hidden layers for the ℓ -th iteration. The first layer is VN_{ℓ} , the second layer is CN_{ℓ} , each of which contains E nodes. The first and the second layers are associated with variable node and check node operations, respectively. The third layer is O_{ℓ} consisting of n nodes. This layer is associated with the hard decision process.

Before drawing edges between nodes in the trellis graph, we have to find culprit edges whose weights, w's, should be trained. We explain this further in an example later. For each edge in the Tanner graph between variable node v_i and check node ch_j , we define two types of edges in the trellis graph, e_{v_i,ch_i} and e_{ch_i,v_i} , respectively, for VN_ℓ and CN_ℓ , where $1 \leq \ell \leq L$. We draw edges in the trellis graph using the following steps:

- If $ch_j \in V_i$, then we connect node e_{v_i,ch_j} of the layer VN_1 and the *i*-th node of the first layer.
- Each node e_{ch_i,v_i} of the layer CN_{ℓ} in the ℓ -th iteration is connected to e_{v_k,ch_j} of the layer VN_ℓ , where $v_k \in CH_j$ and $k \neq i$. For each node, we define a weight w_{ch_i, v_i} .
- Each node e_{v_i,ch_i} of the layer VN_ℓ in the ℓ -th iteration is connected to e_{ch_k,v_i} of the layer $CN_{\ell-1}$, where $ch_k \in V_i$ and $k \neq j$.
- Each node e_{ch_i,v_i} in the layer CN_ℓ is connected to v_i in the layer O_{ℓ} , where $ch_j \in V_i$. There is a weight, w'_{ch_i,v_i} , between the node e_{ch_j,v_i} in the layer CN_ℓ and its corresponding variable node, v_i , in the layer O_ℓ .

Hence, we assigned a pair of weights $(w_{ch_j,v_i}, w'_{ch_j,v_i})$ to the edge e_{ch_j,v_i} ; see the following example. Example 1: Let C be a linear code with parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{6}$$

As we see in Fig. 1, there are five edges in its Tanner graph and four of them are involved in a unique 4-cycle in the graph. We aim to train the weight w for one of these four edges. Suppose the edge corresponding to h_{22} is chosen as a special edge. Hence, the pairs of weights which we consider to train are $(1, w'_{ch_1, v_1})$, $(1, w'_{ch_1, v_2})$, $(1, w'_{ch_1, v_3})$, $(w_{ch_2, v_2}, w'_{ch_2, v_2})$, and $(1, w'_{ch_2, v_3})$ for the corresponding edges e_{ch_1, v_1} , e_{ch_1, v_2} , e_{ch_1,v_3} , e_{ch_2,v_2} , and e_{ch_2,v_3} , respectively. The corresponding trellis graph with 2 iterations is shown in Fig. 2.



Fig. 2. The trellis graph of H in Example 1 with two iterations.

C. Back-propagation Algorithm

The back-propagation algorithm looks for the minimum of the error function in the weight space using the gradient descend method [10]. The combination of weights, which minimizes the error function is the solution of the learning problem. This method requires computation of the derivative of the error function with respect to weights in each iteration step. Therefore, we have to guarantee the continuity and differentiability of the error function. The error function that we use is the cross entropy, which is defined as:

$$\mathcal{L}(\mathbf{o}, \mathbf{c}) = \frac{-1}{n} \sum_{v=1}^{n} c_v \log_2(o_v) + (1 - c_v) \log_2(1 - o_v), \quad (7)$$

where $\mathbf{o} = (o_1, \ldots, o_n)$, $\mathbf{c} = (c_1, \ldots, c_n)$, and o_v and c_v are v-th component of the deep neural network output and the transmitted codeword, respectively. If the transmitted codeword is 0, then $c_v = 0$ for all v.

Moreover, in the back-propagation neural network, a sigmoid function is extensively used because it reduces complications involved during the training phase. The sigmoid function σ is a real function from \mathbb{R} to (0,1) defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. In the deep neural network decoder, each node of the hidden layers in the trellis graph contains two messages which we label them as the left and right messages. In the following, we explain how to obtain these messages for each node of the three hidden layers in the ℓ -th iteration.

• In VN_ℓ layer, the right message $\mu_{v_i,ch_j,\ell}$ of each node is defined as:

$$\tanh\left(\left(\ell_{v} + \sum_{ch_{k} \in V_{i}, \ k \neq j} w_{ch_{k}, v_{i}} \mu_{ch_{k}, v_{i}, (\ell-1)}\right) / 2\right).$$
(8)

The left message of each node is a vector of derivatives $\partial \mu_{v_i,ch_j,\ell}$ of $\mu_{v_i,ch_j,\ell}$ with respect to $\mu_{ch_k,v_i,(\ell-1)}$, $\frac{\partial \mu_{v_i,ch_j,\ell}}{\partial \mu_{ch_k,v_i,(\ell-1)}}$

• In the CN_{ℓ} layer, the right message $\mu_{ch_i, v_i, \ell}$ of each node is

$$2 \tanh^{-1} \left(\prod_{v_k \in CH_j, \ k \neq i} \mu_{v_k, ch_j, \ell} \right).$$
(9)

The left message of each node is a vector of derivatives $\partial \mu_{ch_j,v_{\underline{i}},\ell}$ of μ_{chj},v_{i,ℓ} with respect to μ_{vk},ch_j,ℓ, ^{∂μ_{chj},v_{i,ℓ}}/_{∂μ_{vk},ch_{j,ℓ}}.
In the O_l layer, the right message o_{v,ℓ} of each node is

$$\sigma\left(\ell_v + \sum_{ch_k \in V_i} w'_{ch_k, v_i} \mu_{ch_k, v_i, \ell}\right).$$
(10)

The left message of each node is a vector of derivatives of $o_{v,\ell}$ with respect to $\mu_{ch_k,v_i,\ell}$, $\frac{\partial o_{v,\ell}}{\partial \mu_{ch_k,v_i,\ell}}$.

We call the above constructed trellis graph whose nodes contain messages mentioned above a feed-forward neural network. In fact, such a network represents a chain of function compositions, which transform an input to an output vector. The learning problem consists of finding the optimal combination of weights so that the decoded vector is as close as possible to the transmitted codeword. In other words, we aim to obtain weights by which the following error function \mathcal{L} is minimized:

$$\mathcal{L} = \sum_{\ell=1}^{L} \mathcal{L}(\mathbf{o}_{\ell}, \mathbf{c}),$$

where o_{ℓ} is the output of neural network at ℓ -th iteration, $1 \leq \ell \leq L$. Suppose the vector of weights are W = $(w_1,\ldots,w_{E'},w'_1,\ldots,w'_E)$, where E' < E is the number of culprit edges which are chosen to train their weights. We can minimize \mathcal{L} by using an iterative process of gradient descent for which we need to calculate the gradient $\nabla \mathcal{L}$ as follows

$$\left(\frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_{E'}}, \frac{\partial \mathcal{L}}{\partial w_1'}, \dots, \frac{\partial \mathcal{L}}{\partial w_E'}\right).$$

If $\nabla \mathcal{L} \neq 0$, then we have to update weights and begin the process with new weights. We expect to find a minimum of the error function, when $\nabla \mathcal{L} = 0$. In order to update weights, a learning constant (rate), denoted by α , is used in the gradient descent algorithm. The learning rate is a proportionality parameter, which defines the step length of each iteration in the negative gradient direction. $W_{\text{new}} = W_{\text{old}} - \alpha \cdot \nabla \mathcal{L}$. In the following example we illustrate how to update weights on the trellis graph.

Example 2: Consider the parity-check matrix of Example 1 with L = 2 and the initial input $\mathbf{y} = (-0.5, 2.5, -4)$ (received vector corresponding to sent codeword c = 0) and with the initial weight vector W = (0.1, 0.15, 0.16, 0.17, 0.18, 0.19) in which the first element is associated to w_{ch_2,v_2} and the other elements are associated to w'_{ch_i,v_i} . We also let the training rate be $\alpha = 0.1$. We compute all messages both left and right in the trellis graph. Then, using the paths between the nodes of the last layer and nodes in the CN_1 and CN_2 layers in the trellis graph we update the weights. For example to update w_{ch_2,v_2} , we have to find paths from the last layer to the node e_{ch_2,v_2} of the first iteration. As can be seen in Fig. 2, we



Fig. 3. The comparisons of performances of C1, C3 and C2, C4.

have two paths to this node, which are highlighted bold. The paths P_1 and P_2 are $o_{1,2} \rightarrow e_{ch_1,v_1} \rightarrow e_{v_2ch_1} \rightarrow e_{ch_2,v_2}$ and $o_{3,2} \rightarrow e_{ch_1,v_3} \rightarrow e_{v_2ch_1} \rightarrow e_{ch_2,v_2}$, respectively. Therefore, the derivative of the function \mathcal{L} with respect to w_{ch_2,v_2} is $\frac{\partial \mathcal{L}}{\partial w_{ch_2,v_2}} = -0.013967$. Hence, the new weight for the edge $e_{ch_2,v_2} \text{ is } (w_{ch_2,v_2})_{\text{new}} = (w_{ch_2,v_2})_{\text{old}} - \alpha \times \frac{\partial \mathcal{L}}{\partial w_{ch_2,v_2}} = 0.1 - 0.1 - 0.1 + 0$ $0.1 \times (-0.013967) = 0.101396$. Continuing this process we obtain the new weights W as:

(0.101396, 0.099499, 0.182551, 0.169208, 0.39523, 0.185566).

Comparison between the works of [8] and [9] shows that the former does not perform hard decision till the end of the last iteration while the latter performs hard decision in every iteration. Whereas, when it comes to decoding using SPA we perform hard decision in every iteration. However, we observed that in each variable node operation if instead of adding the initial input to the incoming messages we add the updated LLRs from the previous iteration to the incoming messages, we obtain better performances. To show this, we compare the performance curves of two LDPC codes of sizes 10×15 and 63×105 using initial LLRs, C2, C4, respectively, and using updated LLRs, C1, C3, respectively. The difference between these curves is noticeable; see Fig. 3.

IV. DEEP LEARNING METHOD TO DECODE LATTICES CONSTRUCTED BASED ON CONSTRUCTION A

The belief propagation algorithm obtains poor performance results compared to maximum likelihood algorithms if applied to HDPC codes. Therefore, to achieve better performances, we provide a deep neural network decoder for such lattices constructed based on Construction A. Using the same notation proposed in [11] and the deep neural network decoder presented in Section III, we present the lattice decoding algorithm.

We take the volume-to-noise-ratio as VNR = $\frac{4^{(2n-k)/n}}{2\pi e \sigma^2}$. Suppose we send the all-zero codeword c. We employ BPSK modulation to convert **c** into $(-1, \ldots, -1)$. If the noise **n** is distributed as a normal distribution $\mathcal{N}(0, \sigma^2)$ with zero mean and variance σ^2 , then the transmitted vectors are of the form $\mathbf{y} = \mathbf{c} + 4\mathbf{z} + \mathbf{n}$, [11]. At the first step, \mathbf{z} has to be decoded. We define the decoded \mathbf{z} as $\hat{\mathbf{z}} = \lfloor \frac{\mathbf{y} - (1, \dots, 1)}{4} \rfloor$ as an all-one vector is added during encoding [11]. Now we define $a_i = y_i - 4\hat{z}_i$, for $1 \le i \le n$, and put $\hat{a}_i = 2 - a_i$ for $a_i > 1$ and a_i itself otherwise. Similar to [11], we need to define the *i*-th input LLRs as

$$\ell_i = \frac{(\hat{a}_i + 1)^2}{2\sigma^2} - \frac{(\hat{a}_i - 1)^2}{2\sigma^2}.$$
 (11)

Considering these LLRs as the received word of the deep neural network decoder, we decode \mathbf{c} with $\tilde{\mathbf{c}}$ as its decoded vector. We convert $\tilde{\mathbf{c}}$ to ± 1 notation and call the obtained vector $\tilde{\mathbf{c}}'$. Finally, we let $\hat{c}_i = 2 - \tilde{c}'_i$ if $a_i > 1$ and \tilde{c}'_i otherwise. Then, the decoded lattice vector is $\hat{x} = \hat{c} + \hat{z}$. The stopping criterion in the back-propagation algorithm is $\nabla \mathcal{L} = \mathbf{0}$, in other words we can get the trained weights when the trend of cross entropy changes from downwards to upwards. In practice, meeting this criterion is computationally costly. Instead, we impose a second termination condition. We define a parameter β for the value of cross entropy which depends on VNR. Therefore, the stopping criteria in our neural network decoding algorithm for lattices are either: 1) a change in the trend of the cross entropy, and/or 2) the value of cross entropy becomes smaller than β . Therefore, the two parameters α and β determine the speed and precision of the experiment. The smaller parameters provide a better chance to train weights although, in this case, we need more time to get results. The learning rates chosen in [8] and [9] are $\alpha = 0.001$ and $\alpha = 0.003$, respectively.

Example 3: We applied our proposed neural network decoding algorithm with four iterations to decode Barnes-Wall lattice BW_8 (see [1]). We first determined the culprit edges in the Tanner graph of the Reed-Muller code, which is used in construction of BW_8 . The boldface 1's correspond to culprit edges. We train these edges using back-propagation algorithm to find the best possible weights. For initial weights, we used a vector of length 33 with normal distribution. We chose $\alpha = 0.1$ and $\beta = 0.01$ for VNR = 1. By assuming weight 1 for each edge, the bit error rate (BER) is 0.13 and the trained weights cause an improvement in BER, 0.098. For VNR = 3 and 6, we let $\alpha = 0.1$ and $\beta = 0.001$ and $5 \times (10)^{-5}$, respectively. An error performance comparison for BW₈ between SPA in [11] and our proposed decoding algorithm with four iterations is shown in Fig. 4. It is clear that by reducing β we can have better results for different VNR's.

$$\mathbf{H} = \begin{bmatrix} \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 & 1 & 1 & 1 \\ \mathbf{1} & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \mathbf{1} & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
(12)

The dependency of simulation results to the choices of parameters α , β and culprit edges is the topic of our future work. We also plan to extend neural network decoding algorithms to lattices constructed based on Construction D and D' with underlying HDPC codes such as Reed-Muller codes.



Fig. 4. A comparison between the performance of SPA and back-propagation algorithm.

V. CONCLUSION

In this paper, for the first time, a deep neural network decoding algorithm was presented for Construction A based lattices whose parity-check matrices are high-dense and have high decoding failure rates when iterative algorithms are used to decode them. In contrast with the neural network decoding algorithms in the literature, which were proposed for decoding linear codes and weights for all edges are required to be trained, we focused on training weights for edges which occur in most of 4-cycles and are, potentially, the main culprits of high decoding failure rates. Computer simulations were conducted to compare the error performance of the proposed algorithm versus a message passing algorithm.

REFERENCES

- E.S. Barnes and N.J.A. Sloane, "New lattice packings of spheres," *Canad. J. Math.*, vol. 35, pp. 117–130, 1983.
- [2] J.H. Conway and N.J.A. Sloane, "Sphere Packings, Lattices, and Groups," Springer Verlag, 1998.
- [3] M-R. Sadeghi, A.H. Banihashemi, and D. Panario, "Low-density paritycheck lattices: Construction and decoding analysis," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4481–4495, 2006.
- [4] Y.S. Choi, I.J. Baik, and S.Y. Chung, "Iterative decoding for low-density parity-check lattices," 2008 International Conf. on Adv. Commun. Tech., pp. 358-361, 2008.
- [5] L. Safarnejad and M-R. Sadeghi, "FFT Based Sum-Product Algorithm for LDPC Lattices," *IEEE Commun. Letters*, vol. 16, no. 9, pp. 1504– 1508, 2012.
- [6] G.D. Forney, Jr., M.D. Trott, and S.-Y. Chung, "Sphere-boundachieving coset codes and multilevel coset codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 3, pp. 820–850, 2000.
- [7] E. Viterbo and J.J. Boutros, "A universal lattice decoder for fading channels," *IEEE Trans. Inf. Theory*, vol. 45, pp. 1639–1642, Jul. 1999.
 [8] E. Nachmani, Y. Be'ery and D. Burshtein, "Learning to decode linear
- codes using deep learning," 54th Annual Allerton, pp. 341-346, 2016. [9] E. Nachmani, E. Marciano, L. Lugosch, W.J. Gross, D. Burshtein, and
- [7] E. Nachman, E. Matchalo, E. Edgosch, W.J. Oloss, D. Bushlein, and Y. Be'ery, "Deep Learning Methods for Improved Decoding of Linear Codes", *IEEE J. Selected Topics in Sig. Proc.*, vol. 12, no. 1, pp. 119-131, 2018.
- [10] R. Rajosh, "Neural Networks", Springer-Verlag, Berlin, 1996.
- [11] H. Khodaiemehr, M-R. Sadeghi, and A. Sakzad, "Practical encoder and decoder for power constrained QC-LDPC lattice codes", *IEEE Trans. Commnu.* vol. 65, no. 2, 2017.
- [12] V. Corlay, J.J. Boutros, Ph. Ciblat, and L. Brunel, "Neural Lattice Decoders," arXiv preprint arXiv: 1807.00592, July 2018.