Learning to Decode LDPC Codes with Finite-Alphabet Message Passing

Bane Vasić[1], Xin Xiao[1], and Shu Lin[2]
[1]Dept. of Electrical and Computer Engineering, University of Arizona
[2]Dept. of Electrical and Computer Engineering, University of California, Davis
Email:[1]{vasic, 7xinxiao7}@email.arizona.edu
[2] shulin@ucdavis.edu

***Abstract*—In this paper, we discuss the perspectives of utilizing deep neural networks (DNN) to decode *Low-Density Parity Check* (LDPC) codes. The main idea is to build a neural network to learn and optimize a conventional iterative decoder of LDPC codes. A DNN is based on Tanner graph, and the activation functions emulate message update functions in variable and check nodes. We impose a symmetry on weight matrices which makes it possible to train the DNN on a single codeword and noise realizations only. Based on the trained weights and the bias, we further quantize messages in such DNN-based decoder with 3-bit precision while maintaining no loss in error performance compared to the min-sum algorithm. We use examples to present that the DNN framework can be applied to various code lengths. The simulation results show that, the trained weights and bias make the iterative DNN decoder converge faster and thus achieve higher throughput at the cost of trivial additional decoding complexity.**

I. Introduction

The concept of designing neural networks (NN) with the functionality of a decoding algorithm for error correction codes has been proposed in the early nineties. One type of NNs is proposed for convolutional codes, which adopts *Recurrent Neural Network* (RNN) [1]–[3] to implement the Viterbi algorithm. The other type of NNs is designed for linear block codes [4]–[6]. The common feature of this approach is that decoding is treated as a classification problem, and the NN learns how to classify the channel output words, and thus forms the decision region for each codeword. As a typical classification problem, the training set had to include all codewords in a code space, making the sizes of both training set and neural network exponential in the dimension of code. As a result, these method becomes intractable except for very short codes. Recently, Nachmani *et al.* [7], Lugosch and Gross [8] and Nachmani *et al.* [9] proposed using Deep Neural Networks (DNNs) to improve the Belief Propagation (BP) decoding of High Density Parity Check (HDPC) codes on the *Additive White Gaussian Noise Channel* (AWGNC). These DNNs are constructed based on *Tanner graph*, with various structures including *Multi-Layer Perceptrons Neural Network* (MLPNN) and RNN. One common but key characteristic among them is that the activation functions over hidden layers enforce the equality of weights of a given neuron, which translates to ensuring symmetry of the node update functions. This allows the training to be performed on a single codeword and its noise realizations rather than on the entire code space. The proposed DNN decoders for HDPC codes can be viewed as weighted BP decoders, where the trainable weights and bias are assigned over edges in the Tanner graph of HDPC codes. Applying learning methods such as *Stochastic Gradient Descent* (SGD) and Adam [10] to find the weights and the bias allows such NN decoders to compensate for short cycles (4-cycles) in the Tanner graph of HDPC codes, and to improve the BP decoding performance. In the same spirit, Xu *at al.* [11] used the DNN to improve the BP decoding for Polar codes, while Gruber *at al.* [12] showed that for very short length, decoders of structured codes are indeed easier to learn than that of random codes, and that the training based on all possible codewords results in a NN decoder that has performance approaching maximum a posteriori (MAP) decoding.

In this paper, we propose to use MLPNN to learn and optimize iterative decoders of finite length LDPC codes. Once the weight values are determined, the NN is translated back to a conventional description of variable and check node update functions. Our focus is on the update functions defined on finite-precision messages which lead to *Finite-Alphabet Iterative Decoders* (FAIDs) [13]. The BP decoders with messages quantized using 6-7 bits [14]–[16] has been shown not to suffer significant performance degradation compared to floating BP or BP-based algorithms on the AWGNC. On the other hand, it is know that for the Binary Symmetric Channel (BSC) a FAID with only 3-bit precision outperforms the BP and all other message passing decoders [13], [17]. This is achieved by designing the FAID message update rules that correct trapping sets with a dominant contribution to the error floor. Recently, Nguyen-Ly *at al.* [18] and Meidlinger *at al.* [19] used density evolution to optimize the FAID over the AWGNC. In this paper, we also consider the AWGNC, where traditional iterative decoders take longer to converge to a valid codeword. The goal is to achieve comparable performance with a small maximum number of iterations. More precisely, we utilize NN to optimize a FAID on the AWGNC to achieve a desired trade-off between the error performance and decoding latency. In particular, we are interested in improving the waterfall performance while restricting the maximal number of iterations to a very small value.

A MLPNN is constructed based on Tanner graph of the LDPC code and defines a set of activation functions according

to "proto" message update functions such as min-sum or an exiting FAID. Instead of training different weights over distinct edges in each iteration as in [9], [11], we impose the constraints over weights and biases to direct the learning process, while keeping small number of parameters. This results in a faster learning, lower decoding complexity and memory requirements while preserving good error performance.

The rest of the paper is organized as follows. Section II gives the necessary background. Section III presents the framework of iterative MLPNN decoder. Section IV introduces DE and the quantization based on the trained parameters. Section V demonstrates the examples and simulation results. Section VI concludes the paper.

II. Preliminaries

Let $\mathcal{C}$ be a LDPC code and $\mathcal{G} = (V, C, E)$ be its Tanner graph, where V/C is the set of variable /check nodes, and E is the set of edges. If the code length is N, the number of parity check equations is M, and the number of edges is I, then $|V| = N$, $|C| = M$, and $|E| = I$. Let the i-th variable node be v_i, j-th check node be c_j, then the edge connecting v_i and c_j is denoted by (v_i, c_j), $1 \le i \le N, 1 \le j \le M$.

Let $\mathbf{x} = (x_1, x_2, ...x_N)$ be the transmitted codeword and $\mathbf{y} = (y_1, y_2, ...y_N)$ be the received channel output vector. i.e., $y_i = (-1)^{x_i} + z_i$ for $1 \le i \le N$ where z_i is gaussian noise with standard deviation σ. The likelihood message is defined by the log likelihood ratios (LLR): $\Lambda = (\lambda_1, \lambda_2, ..., \lambda_N)$, where $\lambda_i = \log \frac{\Pr(x_i=0|y_i)}{\Pr(x_i=0|y_i)}$. Let $\mathcal{D}_{\text{proto}}$ be the given conventional iterative decoder (such as bit-flipping (BF), Gallager-B, min-sum algorithm (MSA), sum-product algorithm (SPA), etc.) to be learned and optimized. Suppose that the messages in ℓ-th iteration in $\mathcal{D}_{\text{proto}}$ are computed using the following updating rules: $\nu^{(\ell)}_{v_i \to c_j} = \Phi(y_i, \mathbf{m}_i^{(\ell)})$ and $\mu^{(\ell)}_{c_j \to v_i} = \Psi(\mathbf{n}_j^{(\ell-1)})$, where $\mathbf{m}_i^{(\ell)}$ ($\mathbf{n}_j^{(\ell)}$) denote the incoming messages to a variable node v_i (check node c_j). Let $L_{\max}$ be the maximum number of iterations in $\mathcal{D}_{\text{proto}}$. The proposed iterative MLPNN decoder is defined by a 3-tuple $\mathcal{D}_{\text{nn}} = (\mathcal{G}, \Phi^{(\ell)}_{opt}, \Psi^{(\ell)}_{opt})$, where $\Phi^{(\ell)}_{opt}$ ($\Psi^{(\ell)}_{opt}$) is the optimized decoding rule based on Φ (Ψ) to update the messages passing from variable nodes (check nodes) to check nodes (variable nodes) in the ℓ-th iteration.

The MLPNN consists of one input layer, K hidden layers and one output layer. Let $\mathbf{r}_0$ ($\mathbf{r}_{K+1}$) be the output of input (output) layer, and $\mathbf{r}_k, 1 \le k \le K$ be the output of the k-th hidden layer. In particular, $\mathbf{r}_k = (r_{k,1}, r_{k,2}, ...r_{k,J_k})$, where J_k is the number of neurons in k-th layer, and $r_{k,t}$ is the the output value of the t-th neuron in k-th layer, $0 \le k \le K+1, 1 \le t \le J_k$. For the input and output layer, we have $J_0 = J_{K+1} = N$. The $(k-1)$-th layer and k-th layer are connected by a trainable neuron weight matrix $\mathbf{W}^{(k)}_{J_k \times J_{k-1}}$, and the bias vector in the k-th layer is denoted by $\mathbf{b}^{(k)}$.

III. A MLPNN decoding framework

To obtain the proposed iterative $\mathcal{D}_{\text{nn}}$, we first construct a MLPNN based on $\mathcal{G}$. Our network will be initialized by the knowledge of Φ and Ψ and will perform learning with a given optimality criterion.

A. The MLPNN structure

The constructed network corresponds to an "unwrapped" Tanner graph where every two hidden layers correspond to one iteration in $\mathcal{D}_{\text{proto}}$. The first hidden layer corresponds to the initialization in $\mathcal{D}_{\text{proto}}$. In total, there are $K = 2L_{\max} + 1$ hidden layers in MLPNN. Except for the first hidden layer, the activation functions over odd (even) hidden layers perform in a manner similar to Φ (Ψ). Based on $\mathcal{D}_{\text{proto}}$, this framework includes the following two classes of MLPNNs: (1) edge-based and (2) node based. In an edge-based MLPNN, each neuron in every hidden layer represents the message over an edge in E in a corresponding iteration of $\mathcal{D}_{\text{proto}}$. All hidden layers have the same number of neurons, which is e.

In a node-based MLPNN, there are two types of neurons representing messages over variable and check nodes in the corresponding iteration of $\mathcal{D}_{\text{proto}}$, respectively. Each hidden layer contains only one type of neurons and has size of either N or M.

More specifically, consider an edge-based MLPNN, the activation function in the k-th layer is defined as follows

$$\mathbf{r}_k = \begin{cases} \Phi(\mathbf{b}^{(\frac{k-1}{2})}\mathbf{y}, \mathbf{W}^{(\frac{k+1}{2})}\mathbf{r}_{k-1}), & \text{if } k \text{ is odd,} \\ \Psi(\mathbf{r}_{k-1}), & \text{if } k \text{ is even.} \end{cases} \quad (1)$$

In Eq. 1, $\mathbf{r}_0 = \Lambda$. Furthermore, all nonzero entries in $\mathbf{W}^{(k)}$ are forced to have the same value, i.e., $\mathbf{W}^{(k)}(i,j) = w^{(k)}$ if $\mathbf{W}^{(k)}(i,j)$ is a nonzero entry in $\mathbf{W}^{(k)}$. Since Φ and Ψ in $\mathcal{D}_{\text{proto}}$ satisfy symmetric conditions [13], Eq. (1) preserves them as well.

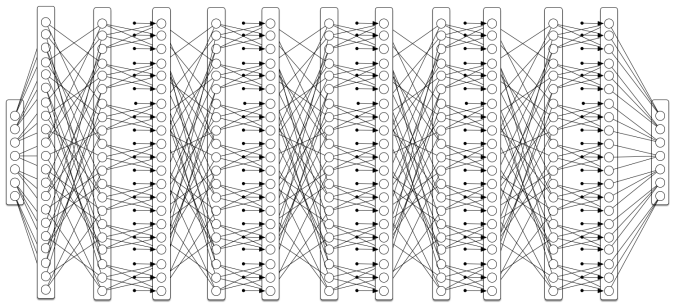


Fig. 1: MLPNN of PG(7,3)

Fig. 1 illustrates the MLPNN structure of the *Projective Geometry* (PG) code PG(7,3). This PG code has code length $N = 7$, rate 0.429, and its parity check matrix has $M = 7$ rows and both column and row weight of 3. We consider $\mathcal{D}_{\text{proto}}$ as the conventional MSA running for maximum of L_{max} iterations, and therefore the corresponding MLPNN is edge-based. All its hidden layers have 21 neurons corresponding to the messages over the 21 edges. The t-th neuron in the first (last) hidden layer is connected to a single input node v_i in the input (output) layer if v_i is incident to edge (t). The rest hidden layers are connected as follows: for an odd (even) hidden layer, if edge $(t) = (v_i, c_j)$, then the t-th neuron

in this layer is connected to all neurons (except t-th neuron) in previous layer whose corresponding edges in Tanner graph are incident to v_i (c_j). The constructed MLPNN consists of 11 hidden layers, which corresponds to $L_{max} = 5$ iterations of MSA. Note that the arrows in the odd hidden layers in Fig. 1 indicate the biased channel values.

B. Learning

Since the channel is output-symmetric, we can assume that the all-zero codeword is transmitted, i.e., $\mathbf{x} = \mathbf{0}$, thus $\mathbf{y} = \mathbf{x} + \mathbf{z} = \mathbf{z}$. With the symmetry conditions on the weight matrices, it is sufficient to use a database composed of the realizations of the noise vector $\mathbf{z} = (z_1, z_2, \ldots, z_N)$. Let $\mathbf{r}_0 = \Lambda$ and $\mathbf{u} = \mathbf{r}_{K+1}$ be the perceptron values in the input and output layer, respectively. Both $\mathbf{r}_0$ and $\mathbf{u}$ have length $J_0 = J_{K+1} = N$, with $\mathbf{r}_0$ receiving likelihood message Λ. When $\mathcal{D}_{\text{proto}}$ is the MSA, instead of using the likelihood message, it is sufficient to feed the noise patterns $\mathbf{z}$ into the MLPNN for learning, i.e., $\mathbf{r}_0 = \mathbf{z}$. This is because the MSA is insensitive to the noise variance. We take account of nonlinear activation functions over $\mathbf{u}$, which convert likelihood messages into probability. A common nonlinear activation function is the sigmoid function $\sigma(x) = \left(1 + x^{-1}\right)^{-1}$. Since $\Pr(x_i = 0|y_i) = \left(1 + e^{-\lambda_i}\right)^{-1} = \sigma(\lambda_i)$ and $\mathbf{u}$ is the estimate of likelihood message Λ, $\sigma(\mathbf{u})$ is the estimate of probability $\Pr(x_i = 0|y_i)$. There are several candidates for loss function as listed in [9], and we consider the following cross-entropy loss function:

$$\mathbf{\Gamma}(\mathbf{u}, \mathbf{x}) = -\frac{1}{N}\sum_{i=1}^{N}(1 - x_i)\log(\sigma(u_i)) + x_i \log(1 - \sigma(u_i)). \tag{2}$$

We use Adam with mini-batches for training. The optimized decoding rules of $\mathcal{D}_{\text{nn}}$ (with floating point precision) are derived based on the trained $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$:

$$\Phi_{opt}^{(\ell)} = \Phi(\widehat{\mathbf{b}}^{(\ell)}\mathbf{y}, \widehat{w}^{(\ell)}\mathbf{m}_i^{(\ell)}), \Psi_{opt}^{(\ell)} = \Psi(\mathbf{n}_j^{(\ell-1)}). \tag{3}$$

Especially, the weight and bias of output layer are used in hard-decision in all iterations.

C. Quantization of neuron outputs in $\mathcal{D}_{\text{nn}}$

Now we explain how to quantize $\mathcal{D}_{\text{nn}}$ based on trained parameters $\widehat{w}^{(\ell)}$ and $\widehat{\mathbf{b}}^{(\ell)}$ for implementation in hardware. A decoder with quantized messages is denoted by $\mathcal{D}_{\text{nn,Q}}$.

Let $\mathcal{A}_{\text{nn,L}_{\max}} = \{-H_l, ..., -H_2, -H_1, 0, H_1, H_2, ...H_l\}$ be the finite message alphabet consisting of $2l + 1$ levels, where $H_i \in \mathbb{R}^+$ and $H_i > H_j$ for any $i > j$. The quantization function $Q(.)$ of messages of $\mathcal{D}_{\text{nn}}$ is defined based on a threshold set $\mathcal{T} = \{T_i : 1 \le i \le l + 1\}$ as follows:

$$Q(x) = \begin{cases} \text{sgn}(x)H_i & \text{if} \quad T_i \le x < T_{i+1} \\ 0 & \text{if} \quad |x| < T_1 \end{cases} \tag{4}$$

where $T_i \in \mathbb{R}^+$ and $T_i > T_j$ for any $i > j$, and $T_{l+1} = \infty$. Conventionally, we can define the quantizer thresholds as: $T_1 = \alpha_1 H_1$, $T_i = \alpha_i H_{i-1} + (1 - \alpha_i)H_i$ for $2 \le i \le l$, based on the set of scalars $\mathcal{S}_l = \{\alpha_1, \alpha_2, ..., \alpha_l\}$ that controls the relative distance between two consecutive levels.

The quantization of the channel output vector $\mathbf{y}$ should maximize the mutual information between channel input and the quantizer output. But for simplicity, the same quantizer is used for messages. The vector $\mathbf{y}$ received from the channel is first quantized into $\mathbf{y}_Q$ using the above rules. When $\mathcal{D}_{\text{nn,Q}}$ starts, $\Psi_{opt}^{(\ell)}$ is the same as that in $\mathcal{D}_{\text{nn}}$ except that messages $\mathbf{n}_j^{(\ell-1)}$ are quantized, and their elements belong to $\mathcal{A}_{\text{nn,L}_{\max}}$. Based on $\{\widehat{w}^{(\ell)}\}$, $\{\widehat{\mathbf{b}}^{(\ell)}\}$ and above rules, for every ℓ-th iteration, the quantized decoder $\mathcal{D}_{\text{nn,Q}}$ performs the following:

$$\Phi_{opt,Q}^{(\ell)} = Q\left(\Phi(\widehat{\mathbf{b}}^{(\ell)}\mathbf{y}_Q, \widehat{w}^{(\ell)}\mathbf{m}_i^{(\ell)})\right). \tag{5}$$

IV. Experiments and numerical results

The simulation of $\mathcal{D}_{\text{nn}}$ with floating point is carried out as $\mathcal{D}_{\text{proto}}$ with additional $\widehat{w}^{(\ell)}$ and $\widehat{\mathbf{b}}^{(\ell)}$, while the test simulation of $\mathcal{D}_{\text{nn,Q}}$ in finite alphabet is carried out with the quantizer. The measure of performance is *bit-error-rate* (BER) or *frame-error-rate* (FER). We compared the performance of $\mathcal{D}_{\text{nn}}$ $\mathcal{D}_{\text{nn,Q}}$, and $\mathcal{D}_{\text{proto}}$ with same $L_{\max}$.

We built MLPNNs in Python3.6 and used Tensorflow [20] library for training. The training set has size of 5000, and the NN is optimized by Adam. The training set consists of realizations of Gaussian noise vectors. We consider the conventional MSA as $\mathcal{D}_{\text{proto}}$, with maximum number of iteration $L_{\max}$ set to 5. In another words, we want to use MLPNN to learn conventional MSA. We constructed two MLPNNs to show the flexibility in terms of code length, one for a short code, Tanner code (155, 64), and the other for a medium length code, QC-LDPC code (1296, 972). For each MLPNN, based on the its trained weights and bias, we further give a finite alphabet $\mathcal{A}_{\text{nn},5}$ of 7 levels and a threshold set $\mathcal{T}$, whose quantization function is equivalently interpreted into a collection of 3-bit variable node update LUTs.

A. Tanner code experiment

The Tanner code has column and row weight of 3 and 5, respectively, thus the MLPNN consists of 11 hidden layers of size $465 = 3 \times 155$. Batch size was set to 500, with varying SNRs = $\{5.5, 6, 6.5, 7, 7.5\}$ and 1000 samples per SNR. The number of epochs is 30. The learning rate of Adam is 0.09, and the trained weights $\{\widehat{w}^{(\ell)}\}$ are

$$\{1.1388, 0.8541, 0.8565, 0.8726, 0.9464, 1.0603, 0.9712\}.$$

The distribution of $\widehat{\mathbf{b}}^{(\ell)}$ in different iterations is shown in Fig. 2. The bias distribution has a high variance at the first two iterations, and it gets narrower as the iteration grows. The variance of bias comes from the training, which has only 30 epochs.

Fig.3 gives the BER performance of $\mathcal{D}_{\text{nn}}$, $\mathcal{D}_{\text{proto}}$, and the normalized MSA (NMSA) with the scaling factor of 0.75. $\mathcal{D}_{\text{nn}}$ outperforms both NMSA and $\mathcal{D}_{\text{proto}}$, and at BER of 10^{-9}, within 5 iterations, the $\mathcal{D}_{\text{nn}}$ achieves coding gain of 0.4 dB over $\mathcal{D}_{\text{proto}}$. The $\mathcal{D}_{\text{nn}}$ with 5 iterations has similar performance

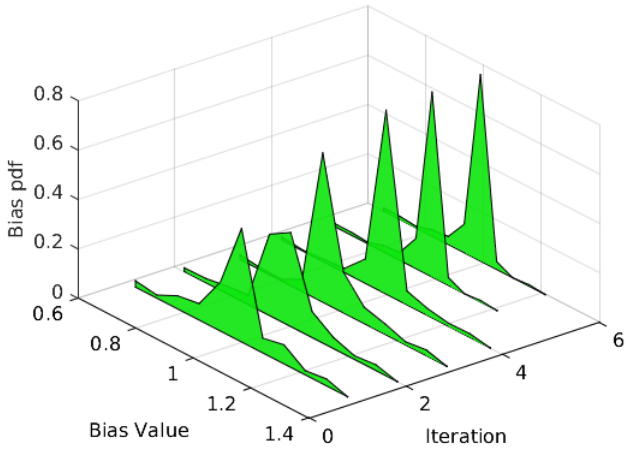


Fig. 2: Bias distribution in different iterations of Tanner code (155,64).

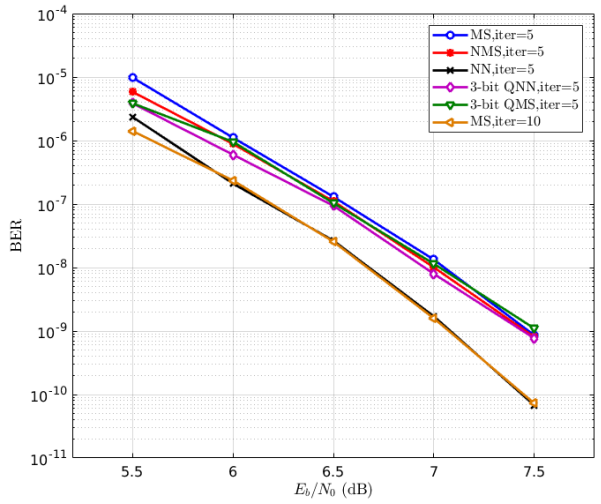


Fig. 3: BER Performance of $\mathcal{D}_{\rm nn}$, $\mathcal{D}_{\rm nn,Q}$, $\mathcal{D}_{\rm proto}$ and $\mathcal{D}_{\rm proto,Q}$ of Tanner code (155,64).

with $\mathcal{D}_{\rm proto}$ with 10 iterations. In other words, the $\mathcal{D}_{\rm nn}$ can achieve faster convergence than $\mathcal{D}_{\rm proto}$.

Based on $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$, we obtain a 7-level $\mathcal{A}_{\rm nn,5}$ and a set of scalars $\mathcal{S}_3$. In particular, $\mathcal{A}_{\rm nn,5} = \{-1.5, -0.8, -0.35, 0, 0.35, 0.8, 1.5\}$ and $\mathcal{S}_3 = \{0.4, 0.4, 0.2\}$. This results in a 7-level quantizer $Q(.)$, i.e., messages are in 3-bit precision. Quantizing both $\mathcal{D}_{\rm nn}$ and $\mathcal{D}_{\rm proto}$ by $Q(.)$, we obtain a 3-bit precision NN decoder $\mathcal{D}_{\rm nn,Q}$ and 3-bit precision conventional MSA $\mathcal{D}_{\rm proto,Q}$. Their performance with 5 iterations are given in Fig.3 as well. The simulation results show that with 5 iterations, $\mathcal{D}_{\rm nn,Q}$ outperforms NMSA, conventional MSA, and $\mathcal{D}_{\rm proto,Q}$. The improvement of both $\mathcal{D}_{\rm nn}$ and $\mathcal{D}_{\rm nn,Q}$ comes from the additional $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$. Equivalently, $\Phi^{(\ell)}_{opt,Q}$ in different iterations can be mapped into a collection of 3-dimensional look up tables (LUTs), which describe the

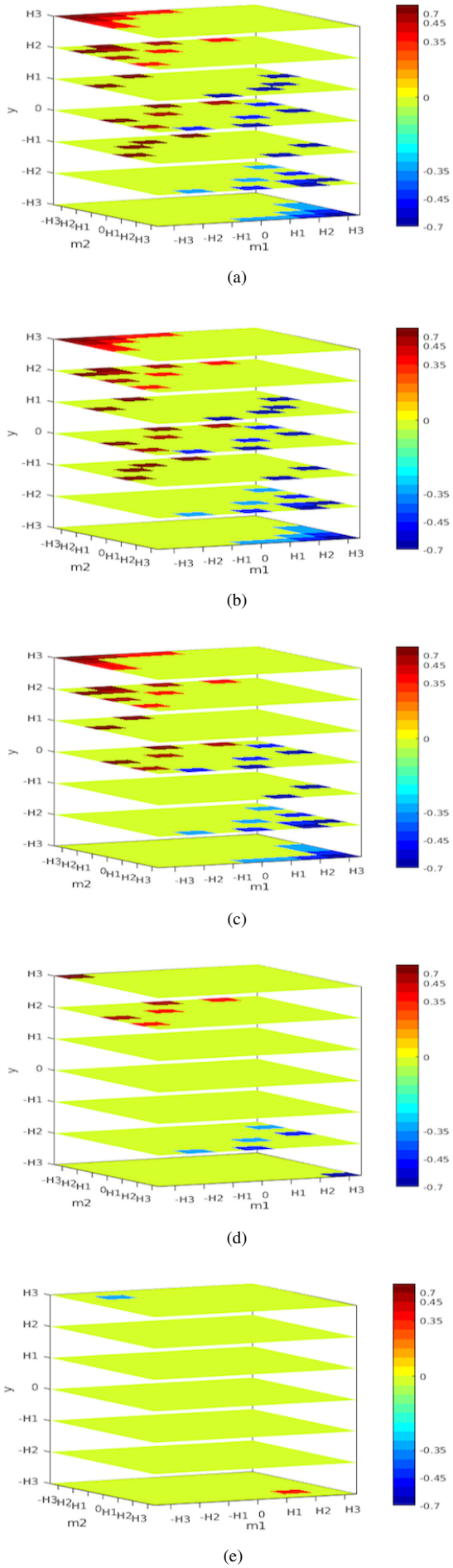


Fig. 4: $\Phi^{(\ell)}_{diff}$ in different iterations of Tanner code (155,64).

variable node updating rules. Based on $\Phi^{(\ell)}_{opt,Q}$, we obtain four distinct LUTs, with the first and second iteration sharing the same LUT. These four distinct LUTs only differ in a few entries. One approach to illustrate how the quantized MLPNN decoder benefits from $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$ is to observe how these LUTs change with iterantion number and how they are different from $\mathcal{D}_{\text{proto,Q}}$. The difference table between LUTs corresponding to $\mathcal{D}_{\text{nn,Q}}$ and $\mathcal{D}_{\text{proto,Q}}$ in the ℓ-th iteration is defined by:

$$\Phi^{(\ell)}_{diff} = \Phi^{(\ell)}_{opt,Q} - \Phi_Q,$$

where $\Phi_Q = Q\left(\Phi(\cdot,\cdot)\right)$. $\Phi^{(\ell)}_{diff}$ of degree-3 Tanner code in different iterations have size of $7 \times 7 \times 7$, and are shown in Fig 4, where the x, y-axes represent the two incoming messages from the check node neighbors and the z axis indicates the channel value. In the first two iterations, $\mathcal{D}_{\text{nn,Q}}$ is very different from $\mathcal{D}_{\text{proto,Q}}$ due to the $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$. As the iteration grows, the difference between LUTs becomes smaller. In the 5-th iteration, $\mathcal{D}_{\text{nn,Q}}$ and $\mathcal{D}_{\text{proto,Q}}$ are almost the same. More specifically, $\Phi^{(\ell)}_{diff}$ tells us how $\mathcal{D}_{\text{nn,Q}}$ improves decoding from $\mathcal{D}_{\text{proto,Q}}$. For example, at the first two iterations, when the channel value is $-H_3$ (corresponding to the bottom flat) and $m_1 = m_2 = H_3$, the output of $\mathcal{D}_{\text{nn,Q}}$ is less than that of $\mathcal{D}_{\text{proto,Q}}$ (which is $H_3 + H_3 - H_3 = H_3$), meaning that $\mathcal{D}_{\text{nn,Q}}$ attenuates the magnitude of the likelihood messages, thus preventing its fast growth. Similar attenuation behavior can be observed in the case of the channel value is H_3 (corresponding to the top flat) and $m_1 = m_2 = -H_3$, when the output of $\mathcal{D}_{\text{nn,Q}}$ is greater than that of $\mathcal{D}_{\text{proto,Q}}$ (which is $-(H_3 + H_3) + H_3 = -H_3$). As the LUTs of $\mathcal{D}_{\text{nn,Q}}$ and $\mathcal{D}_{\text{proto,Q}}$ almost merge in the end, their BER curves do not diverge far away from each other. We note that the similar effect of message attenuation is observed in FAID for the BSC, although these decoders are derived using completely different methodology that relies on trapping set harmfulness [17].

B. Column-weight 4, medium-length code experiment

In the second experiment, we consider the QC LDPC code (1296, 972) of column and row weights equal 4 and 16, respectively. The MLPNN of this code consists of 11 hidden layers of size $5184 = 4 \times 1296$. The batch size was set to 300, with one SNR = 4.5dB and 5000 samples for this SNR. The number of epochs is 100. The learning rate of Adam is 0.001, and the trained weights $\{\widehat{w}^{(\ell)}\}$ are:

$$\{0.9755, 0.7316, 0.7664, 0.7790, 0.7799, 0.7791, 0.7801\}.$$

The distribution of $\widehat{\mathbf{b}}^{(\ell)}$ in different iterations is shown in Fig. 5. Similar to the Tanner code, the variance of bias distribution becomes smaller as the iteration number grows. Since the training for this MLPNN has 100 epochs, the variance becomes very small.

The BER curves of $\mathcal{D}_{\text{nn}}$, $\mathcal{D}_{\text{proto}}$ and normalized MSA with scalar of 0.75 is given in Fig.6. Simulation results show that $\mathcal{D}_{\text{nn}}$ outperforms by 0.3dB the NMSA and by 0.45dB the $\mathcal{D}_{\text{proto}}$ at BER of 10^{-8} within 5 iterations. The $\mathcal{D}_{\text{nn}}$ with 5 iterations performs better than $\mathcal{D}_{\text{proto}}$ with 10 iterations, i.e., the $\mathcal{D}_{\text{nn}}$ converges faster than $\mathcal{D}_{\text{proto}}$.

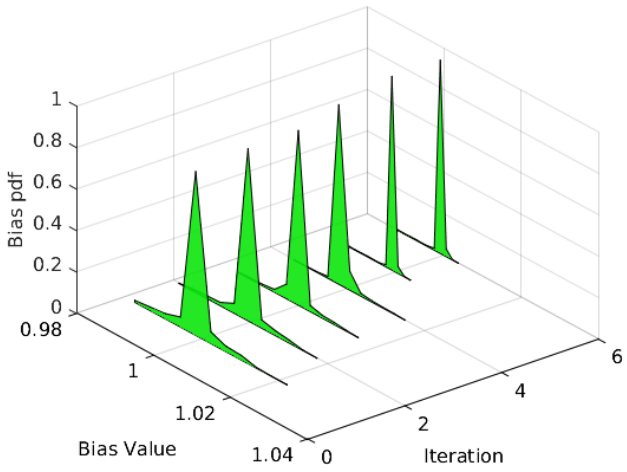


Fig. 5: Bias distribution in different iterations of QC LDPC code (1296, 972).

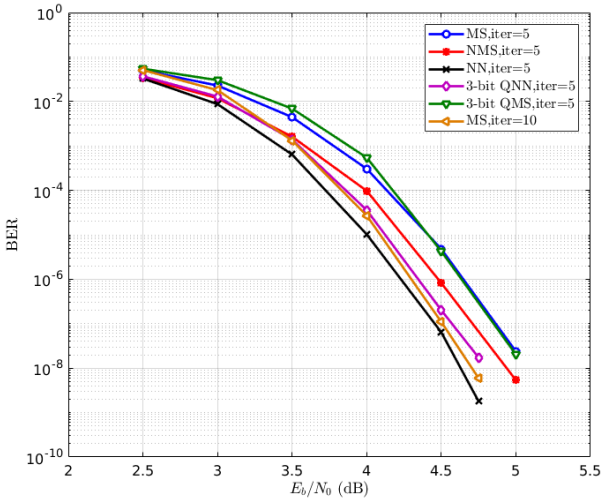


Fig. 6: BER Performance of $\mathcal{D}_{\text{nn}}$, $\mathcal{D}_{\text{nn,Q}}$, $\mathcal{D}_{\text{proto}}$ and $\mathcal{D}_{\text{proto,Q}}$ of QC LDPC code (1296, 972).

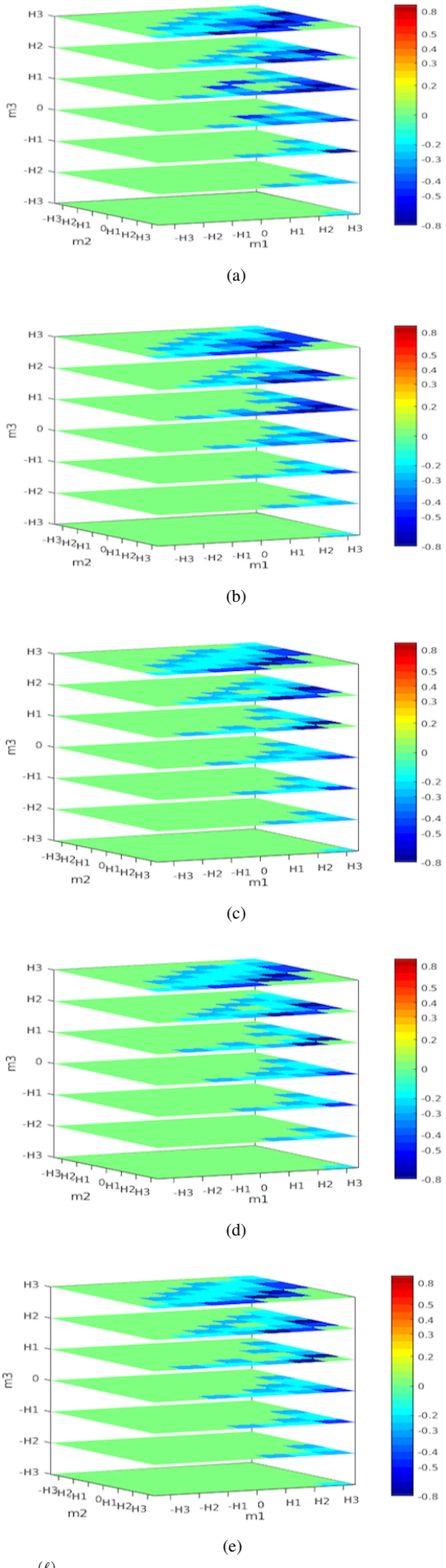
Based on $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$, we obtain a 7-level $\mathcal{A}_{\text{nn},5}$ and a set of scalars $\mathcal{S}_3$. In particular, $\mathcal{A}_{\text{nn},5} = \{-1.0, -0.5, -0.2, 0, 0.2, 0.5, 1.0\}$ and $\mathcal{S}_3 = \{0.4, 0.5, 0.6\}$. This results in a 7-level quantizer $Q(.)$, i.e., messages have 3-bit precision. Quantizing both $\mathcal{D}_{\text{nn}}$ and $\mathcal{D}_{\text{proto}}$ by $Q(.)$, we obtain a 3-bit precision NN decoder $\mathcal{D}_{\text{nn,Q}}$ and 3-bit precision conventional MSA $\mathcal{D}_{\text{proto,Q}}$. Their BER curves with 5 iterations are present in Fig.6 as well. The simulation results show that with 5 iterations, $\mathcal{D}_{\text{nn,Q}}$ outperforms NMSA by 0.13dB and conventional MSA by 0.3dB at BER of 10^{-8}, and it can achieve 0.3 dB coding gain over $\mathcal{D}_{\text{proto,Q}}$. The improvement of both $\mathcal{D}_{\text{nn}}$ and $\mathcal{D}_{\text{nn,Q}}$ again comes from the additional $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$.

Similarly, $\Phi_{opt,Q}^{(\ell)}$ in different iterations can be mapped into five 4-dimensional LUTs. Based on $\Phi_{opt,Q}^{(\ell)}$, we obtain 3 different LUTs, with the last 3 iterations having the same LUT. These 3 distinct LUTs only differ in a few entries. For channel value of $-H_3$, $\Phi_{diff}^{(\ell)}$ of degree-4 QC LDPC code (1296, 972) in different iterations are shown in Fig 7, where the x, y, z-axes represent the three incoming messages from CN neighbors. For all 5 iterations, $\mathcal{D}_{\mathrm{nn,Q}}$ update rules are different from that of $\mathcal{D}_{\mathrm{proto,Q}}$. Again, $\Phi_{diff}^{(\ell)}$ tells us how $\mathcal{D}_{\mathrm{nn,Q}}$ improves decoding compared to $\mathcal{D}_{\mathrm{proto,Q}}$. For example, in the last three iterations, when $m_3 = H_3$ (corresponding to the top flat) and $m_1 = m_2 = H_1$, the output of $\mathcal{D}_{\mathrm{nn,Q}}$ is less than that of $\mathcal{D}_{\mathrm{proto,Q}}$ (which is $H_1 + H_1 + H_3 - H_3 = 2H_1$), meaning that $\mathcal{D}_{\mathrm{nn,Q}}$ again attenuates the message magnitudes. The quantization function in (4) is symmetric, thus all of $\Phi_{diff}^{(\ell)}$, $\Phi_{opt,Q}^{(\ell)}$, Φ_Q satisfy symmetry condition. The "mirror" of $\Phi_{diff}^{(\ell)}$ with channel value of $-H_3$ in 5 iterations is given in Fig 8, whose channel value is $+H_3$. Similarly, in the last three iterations, when $m_3 = -H_3$ (corresponding to the bottom flat) and $m_1 = m_2 = -H_1$, the output of $\mathcal{D}_{\mathrm{nn,Q}}$ is greater than that of $\mathcal{D}_{\mathrm{proto,Q}}$ (which is $-H_1 - H_1 - H_3 + H_3 = -2H_1$), showing that $\mathcal{D}_{\mathrm{nn,Q}}$ attenuates the message magnitudes. $\Phi_{diff}^{(\ell)}$ is almost stable in all iterations, which results in the BER performance difference between $\mathcal{D}_{\mathrm{nn,Q}}$ and $\mathcal{D}_{\mathrm{proto,Q}}$. This $\Phi_{diff}^{(\ell)}$ comes from $\{\widehat{w}^{(\ell)}\}$ and $\{\widehat{\mathbf{b}}^{(\ell)}\}$, resulting in an improved error performance. As we have shown in [21], a decoder supporting multiple decoding rules can be efficiently implemented in hardware, thus an iteration-dependent decoding rule requires only a small hardware overhead.

For $\mathcal{D}_{\mathrm{nn}}$ with floating point, since the training can be conducted offline, the increased decoding computational complexity comes from the additional floating-point multiplications, which is just $2n$ per iteration. For $\mathcal{D}_{\mathrm{nn,Q}}$ with finite precision, especially in 3-bit precision, the memory as well as the computational complexity can be significantly reduced.

V. Conclusion

In this paper, we explore a potential of MLPNN to learn a finite-alphabet iterative message-passing decoding rule of LDPC codes. In the training, we impose additional constraints on weight matrix to control the direction of weight changes and accelerate training. Based on the trained weights and bias, we further quantize the MLPNN decoder messages to 3-bit precision. Examples and simulation results show that within 5 iterations, the MLPNN decoder can achieve at least 0.4 dB over conventional floating point MSA at trivial increase of decoding complexity, and perform no worse than conventional MSA with 10 iterations. Furthermore, the quantized MLPNN decoder with 3-bit precision performs better than both NMSA and conventional MSA. We use the difference between the decoder's LUTs to show how the quantized MLPNN decoder benefits from the trained weights and bias and how it improves decoding compared to the quantized MSA. Numerous open questions are left for future research, such as the dependence



(a)

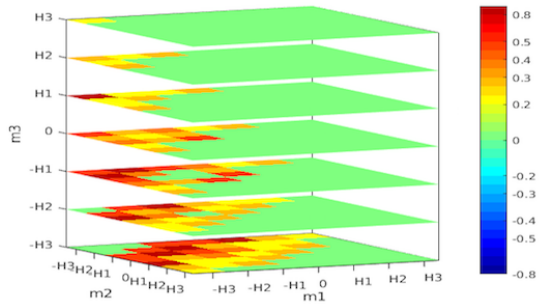
(b)

(c)

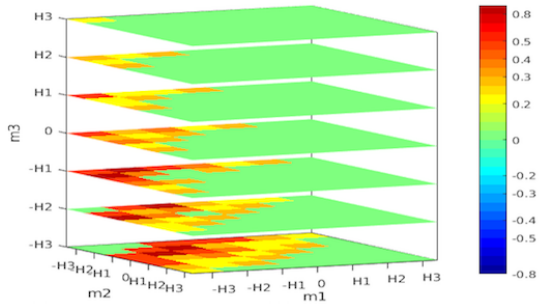
(d)

(e)

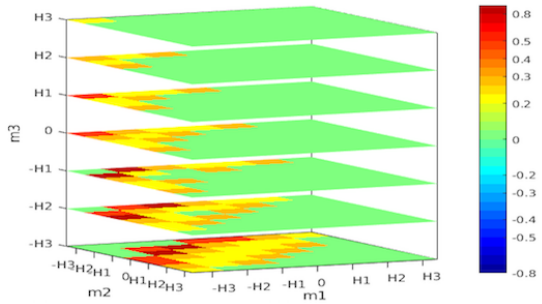
Fig. 7: $\Phi_{diff}^{(\ell)}$ in different iterations of QC LDPC code (1296,972) with channel output $-H_3$.



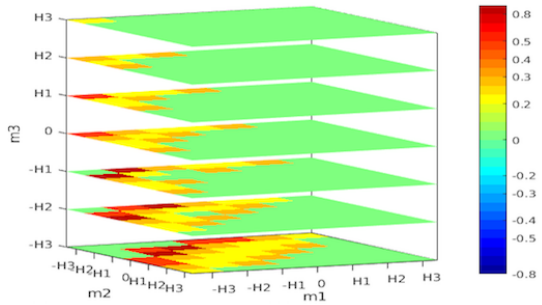
(a)



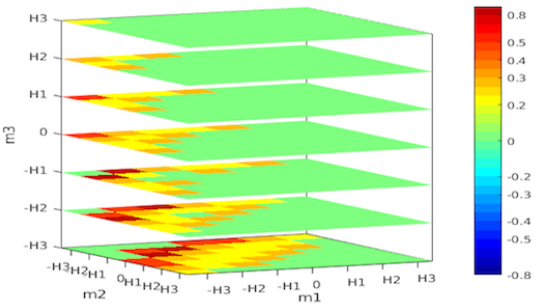
(b)



(c)



(d)



(e)

Fig. 8: $\Phi_{diff}^{(\ell)}$ in different iterations of QC LDPC code (1296,972) with channel output H_3.

of weight matrices on the code structure, effects of number of iterations and quantizer optimization.

ACKNOWLEDGMENT

This work is funded by the NSF under grant NSF ECCS-1500170 and is supported in part by the Indo-US Science and Technology Forum (IUSSTF) through the Joint Networked Center for Data Storage Research (JC-16-2014-US).

REFERENCES

[1] A. Hamalainen and J. Henriksson, "Convolutional decoding using recurrent neural networks," in *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, vol. 5, 1999, pp. 3323–3327.

[2] S. M. Berber and V. Kecman, "Convolutional decoders based on artificial neural networks," in *2004 IEEE International Joint Conference on Neural Networks*, vol. 2, July 2004, pp. 1551–1556 vol.2.

[3] P. J. Secker, S. M. Berber, and Z. A. Salcic, "A generalised framework for convolutional decoding using a recurrent neural network," in *Fourth International Conference on Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint*, vol. 3, Dec 2003, pp. 1502–1506.

[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, Sept. 1989.

[5] Y.-H. Tseng and J.-L. Wu, "High-order perceptrons for decoding error-correcting codes," in *Neural Networks, 1992. IJCNN., International Joint Conference on*, vol. 3, Jun 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, pp. 235–256, 09 2002.

[7] E. Nachmani, Y. Beéry, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference*, 2016, pp. 341–346.

[8] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," 2017. [Online]. Available: https://arxiv.org/abs/1701.05931

[9] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Beéry, "Deep learning methods for improved decoding of linear codes," 2017. [Online]. Available: https://arxiv.org/abs/1706.07043

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[11] W. Xu, Z. Wu, Y. L. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct 2017, pp. 1–6.

[12] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," *CoRR*, vol. abs/1701.07738, 2017. [Online]. Available: http://arxiv.org/abs/1701.07738

[13] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasić, "Finite alphabet iterative decoders, Part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4033–4045, Nov. 2013.

[14] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[15] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (ldpc) codes," *IEEE Transactions on Communications*, vol. 53, no. 4, pp. 549–554, April 2005.

[16] X. Zhang and P. H. Siegel, "Quantized iterative message passing decoders with low error floor for ldpc codes," *IEEE Transactions on Communications*, vol. 62, no. 1, pp. 1–14, January 2014.

[17] D. Declercq, B. Vasić, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders, Part II: Improved guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4046–4057, Nov. 2013.

[18] T. T. Nguyen-Ly, V. Savin, K. Le, D. Declercq, F. Ghaffari, and O. Boncalo, "Analysis and design of cost-effective, high-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–14, Dec 2017.

[19] M. Meidlinger, A. Balatsoukas-Stimming, A. Burg, and G. Matz, "Quantized message passing for LDPC codes," in *2015 49th Asilomar Conference on Signals, Systems and Computers*, Nov 2015, pp. 1606–1610.

[20] M. Abadi, P. Barham, J. Chen, and *et al*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[21] F. Cai, X. Zhang, D. Declercq, B. Vasić, and S. K. Planjery, "Low-complexity finite alphabet iterative decoders for LDPC codes," *IEEE Transactions on Circuits and Systems - Part I: Regular Papers*, 2014.