Finite alphabet iterative decoding of LDPC codes with coarsely quantized neural networks

Xin Xiao, Bane Vasić, and Ravi Tandon
The University of Arizona
Email: {7xinxiao7,vasic,tandonr}@email.arizona.edu

Shu Lin
University of California, Davis
Email: shulin@ucdavis.edu

***Abstract*—In this paper, we introduce a method of using quantized neural networks (QNN) to design finite alphabet message passing decoders (FAID) for *Low-Density Parity Check* (LDPC) codes. Specifically, we construct a neural network with low precision activations to optimize a FAID over Additive White Gaussian Noise Channel (AWGNC). The low precision activations cause a critical issue that their gradients vanish almost everywhere, making it difficult to use classical backward propagation. We introduce *straight-through estimators* (STE) [1] to avoid this problem, by replacing zero derivatives of quantized activations with surrogate gradients in the chain rules. We present a systematic approach to train such networks while minimizing the bit error rate, which is a widely used and accurate metric to measure the performance of iterative decoders. Examples and simulations show that by training a QNN, a FAID with 3-bit of message and 4-bit of channel output can be obtained, which performs better than the more complex floating-point min-sum decoding algorithm. This methodology is promising in the sense that it facilitates designing low-precision FAID for LDPC codes while maintaining good error performance in a flexible and efficient manner.**

I. Introduction

Motivated by successes of *Deep Neural Networks* (DNN) in various applications, there has been a renewed interest in using neural networks (NN) for data detection and decoding, and in the past two years a number of interesting results have been reported. The NN approach has been shown to handle well the scenarios of unknown, time variant, nonlinear, and channels with memory and correlated noise (see [2], [3] and references therein). In the known channel case, the efforts have mainly focused on learning decoding algorithms for error-correction codes (ECC). Recently, several research groups have shown that DNNs can be used to efficiently learn various decoding algorithms, including *Belief Propagation* (BP) decoding [4]–[8].The reason for this effectiveness of DNNs, for instance, in the case of BP decoding based on Tanner graphs, is that the synaptic weight matrices over hidden layers are constrained to preserve the message-passing update rule symmetry conditions. This allows the training to be performed on a single codeword and its noise realizations rather than on the entire code space, and therefore opens up the possibility of using DNN on long codes. In our preliminary work [9], we have shown that DNNs can improve the decoding convergence speed (number of iterations to achieve a desired frame error rate) of conventional *Min-Sum* (MS) decoding algorithm. Unlike well established theories on *Density Evolution* (DE) and *Trapping Set* (TS), which are used to guide decoder design for the waterfall and error-floor regions, respectively, there is no existing theory for speeding up decoding convergence. The multi-layer structure of DNNs can naturally learn time-varying update rules that can provably outperform fixed update rules, thereby, opening new decoding design possibilities.

A crucial drawback of the above approaches is that the DNNs typically used for decoding use synaptic weights with floating point precision which is prohibitively complex for most potential applications. On the other hand, decoders in which messages belong to a finite alphabet are not only of great theoretical, but also practical importance. However, the main challenge in training Quantized NN (QNN) is that low precision weights or activations cause a critical issue that their gradients vanish almost everywhere, making backward propagation inapplicable.

In this paper, we propose to use QNNs to design FAIDs for LDPC codes. To solve the challenge discussed above, we rely on very recent results on training QNNs which is of great interest on its own [10]–[13]. The main techniques to quantize weights and activations include *straight-through estimators* (STE), vector quantization and weight optimization, among which we focus on STE, for the reason that it handles the zero derivatives in the backward propagation. In brief, we are interested in FAIDs over AWGNC. The QNN is an "unwrapped" Tanner graph, with a set of activations defined by "proto" message update functions such as MS decoding. The quantized activations and channel messages are used in both forward and backward propagation. As a consequence, the neural network has full-precision weights, biases and gradients, while it has low precision activations and channel messages. This does not increase decoding complexity, as training such QNN is offline. Based on the trained weights and biases, we obtain a set of Look Up Tables (LUT) to describe time-varying FAID message update rules in each iteration. We use bit error rate as the objective function, since it is a more common and accurate metric to measure the performance of iterative decoders. More importantly, we choose two proper STEs as the proxy derivatives in the back propagation for the quantizers of activations and BER objective function. The simulation results show that the trained FAIDs with only 3-bit messages and 4-bit channel outputs outperform floating MS decoder, which is desirable in hardware implementation. This methodology provides a systematic design of good FAIDs for

LDPC codes, with any reasonable quantization scheme and varying code lengths, thus it is very flexible and simple. To the best of the knowledge, this is the first work to use QNNs with STE to design FAIDs of LDPC codes.

The rest of the paper is organized as follows. Section II gives the necessary background. Section III presents the QNN framework. Section IV introduces the training with QNN. Section V demonstrates the examples and simulation results. Section VI concludes the paper.

II. Preliminaries

Let $\mathcal{C}$ be a LDPC code, $\mathbf{H}$ be its parity check matrix, and $\mathcal{G} = (V, C, E)$ be its Tanner graph, where V (respectively, C) is the set of variable (respectively, 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 $|E| = I$. Denote the i-th variable node as v_i, j-th check node as c_j, and the edge connecting v_i and c_j as (v_i, c_j) which is indexed by some integer (e), $1 \le i \le n, 1 \le j \le m, 1 \le e \le I$. Let the d_{v_i} be the degree of v_i, and d_{c_j} be the degree of c_j.

Let $\mathbf{x} = (x_1, x_2, ...x_n) \in \mathcal{C}$ and $\mathbf{y} = (y_1, y_2, ...y_n)$ be the transmitted codeword and received channel output vector, respectively. Consider binary phase shift keying (BPSK) modulation, 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 $\lambda_i = \log \frac{\Pr(x_i=0|y_i)}{\Pr(x_i=1|y_i)}$. Suppose that the message update rules of variable nodes and check nodes which the QNN aims to optimize are defined as $\nu_{v_i \to c_j} = \Phi(\lambda_i, \mathbf{p}_i)$ and $\mu_{c_j \to v_i} = \Psi(\mathbf{q}_j)$, respectively, where $\mathbf{p}_i$ ($\mathbf{q}_j$) is the incoming message to a variable node v_i (check node c_j). $\tilde{\lambda}_i = \Upsilon(\lambda_i, \mathbf{p}_i)$ denotes the likelihood message approximation for variable node v_i, which will be optimized by QNN as well. Let Q^{msg} and Q^c be the quantizers of messages and channel outputs (or likelihood messages), with b^{msg}-bit and b^c-bit precision, respectively. The corresponding alphabet sets are denoted by $\mathcal{A}^{msg}$ and $\mathcal{A}^c$, and quantization threshold sets are denoted by $\mathcal{T}^{msg}$, $\mathcal{T}^c$.

III. The QNN frameworks

A. Low precision QNN activations

The proposed QNN is constructed based on Tanner graph $\mathcal{G}$, with three types of activations defined by Φ, Ψ, and Υ. In general, the message update rules Φ, Ψ, and Υ can be provided by any conventional iterative decoder such as sum-product decoder, bit-flipping decoder, etc. In this work, we take Φ, Ψ, and Υ from MS decoder, and then use two quantizers Q^{msg} and Q^c to quantize the activations and the likelihood messages (or the channel outputs), respectively. More specifically, suppose that the QNN consists of K hidden layers, with output values denoted by $\mathbf{r}^{(k)}, 1 \le k \le K$. $\mathbf{r}^{(0)}$ ($\mathbf{r}^{(K+1)}$) represents the values of input (output) layer. $\mathbf{r}^{(k)} = (r_1^{(k)}, r_2^{(k)}, ...r_{J_k}^{(k)})^T$, where J_k is the number of neurons in k-th layer, and $r_t^{(k)}$ is the the output value of the t-th neuron

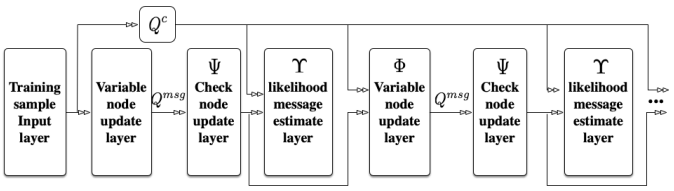

Fig. 1. Block diagram of a QNN.

in k-th layer, $0 \le k \le K+1, 1 \le t \le J_k$. Fig. 1 shows the block diagram of a QNN, where it is readily observed that every three hidden layers correspond to one iteration. $\mathbf{r}^{(k)}$ represents the variable (respectively, check) nodes message update if $3|k-1$ (respectively, $3|k-2$), and it represents the estimate of likelihood messages if $3|k$. Each neuron in variable and check nodes message update layers stands for an edge. Each neuron in likelihood message approximation layers stands for a variable node. Therefore, we have $J_k = n$ if $3|k$, otherwise $J_k = I$. The weight matrix and the bias vector in the k-th layer are denoted by $\mathbf{W}^{(k)}$ and $\mathbf{b}^{(k)}$, respectively.

The output of the k-th hidden layer ($k > 1$) is computed as follows:

$$\mathbf{r}^{(k)} = \begin{cases} Q^{msg}(\Phi(\mathbf{b}^{(k)} Q^c(\Lambda), \mathbf{W}^{(k)} \mathbf{r}^{(k-2)})), & 3|(k-1), \\ \Psi(\mathbf{r}^{(k-1)}), & 3|(k-2), \\ \Upsilon(\mathbf{b}^{(k)} Q^c(\Lambda), \mathbf{W}^{(k)} \mathbf{r}^{(k-1)}), & 3|k, \end{cases} \quad (1)$$

where for any $(e) = (v_i, c_j)$, if $3|(k-1)$,

$$r_e^{(k)} = Q^{msg}(b_e^{(k)} Q^c(\lambda_i) + \sum_{(e')=(v_i,c_t), t \ne j} w_{e,e'}^{(k)} r_{e'}^{(k-2)}),$$

if $3|(k-2)$,

$$r_e^{(k)} = \prod_{(e')=(v_t,c_j), t \ne i} \operatorname{sgn}(r_{e'}^{(k-1)}) \cdot \min_{(e')=(v_t,c_j), t \ne i} \left| r_{e'}^{(k-1)} \right|,$$

and if $3|k$,

$$r_i^{(k)} = b_i^{(k)} Q^c(\lambda_i) + \sum_{(e')=(v_i,c_t)} w_{i,e'}^{(k)} r_{e'}^{(k-1)}.$$

In this work, we consider that Q^c and Q^{msg} are predefined. At each k-th layer where $3|k$, for each training sample, we check whether its estimated codeword satisfies all parity equations or not. If so, this training sample will skip the rest layers and its output at the current k-th layer will be directly used to calculate the objective function, namely,

$$\mathbf{r}^{(K+1)} = \mathbf{r}^{(k)}, \text{ if } \frac{\mathbf{1} - \operatorname{sgn}((\mathbf{r}^{(k)})^T)}{2} \cdot \mathbf{H}^T = \mathbf{0}. \quad (2)$$

The first hidden layer is the initialization of the QNN from channel, and its output is calculated by

$$\mathbf{r}^{(1)} = Q^{msg}(\mathbf{W}^{(1)} \mathbf{r}^{(0)}), \quad (3)$$

where $\mathbf{r}^{(0)} = \Lambda$, $r_e^{(1)} = Q^{msg}(w_{e,i}^{(1)} r_i^{(0)}), \forall (e) = (v_i, c_j)$. Since Ψ is the min operation which is insensitive to the noise variance, we replace all likelihood messages λ_i by channel outputs y_i in Eq. (1) and (3), and thus Q^c is the quantizer of $\mathbf{y}$. As quantizers play roles of nonlinear functions, all the layers have nonlinear activations.

B. Flexible quantizers of messages and channel outputs

The quantizers Q^s in Eq. (1) of b^s-bit precision is defined by an alphabet set $\mathcal{A}^s$ and a threshold set $\mathcal{T}^s$ as follows, where the superscript "s" stands for either "*msg*" or "*c*": for given finite sets $\mathcal{A}^s = \{0, \pm L_1^s, \pm L_2^s ..., \pm L_{2^{b^s-1}-1}^s\}$ and $\mathcal{T}^s = \{T_1^s, T_2^s ..., T_{2^{b^s-1}-1}^s, T_{2^{b^s-1}}^s = \infty\}$, with $L_i^s, T_i^s \in \mathbb{R}^+, 1 \leq i \leq 2^{b^s-1}-1$ and $L_i^s > L_j^s, T_i^s > T_j^s$ for any $i > j$,

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

From Eq. (4), it is readily observed that Q^s is symmetric. Since Φ, Ψ, and Υ satisfy symmetry conditions, Eq. (1) preserves them as well. In addition, when passing messages and channel values from one layer to another in the forward propagation, $\mathcal{A}^s$ can be first mapped into integers to save memories, and only integers with b^{msg}-bit (or b^c-bit) precisions are passed instead of values in $\mathcal{A}^s$. When computing the quantized activations, integers are mapped back into $\mathcal{A}^s$.

In general, for AWGNC, the precision of channel outputs is higher than that of messages, i.e., $b^{msg} < b^c$. Different methods are used in literatures to design a good Q^c. For example, a Q^c can be derived by maximizing the mutual information between channel input and the quantizer output [14], or by Lloyd-Max scalar quantization method [15], [16] to minimize the mean square error (MSE) between the quantized and nonquantized densities. In this work, we maximize the mutual information between channel input and the quantizer output to design Q^c. Since the mutual-information based approach receives the noise deviation σ, code rate, and b^c as parameters, for given precision and LDPC code, there are several plausible Q^c's that can be obtained, depending on the choice of σ. The design of Q^{msg} is in itself an interesting problem. One way we could use to construct a Q^{msg} is based on Q^c. In particular, we take a symmetric subset of $\mathcal{A}^c$ to form $\mathcal{A}^{msg}$ and use a set of scalars $\mathcal{S}^{msg} = \{\alpha_1, \alpha_2, ..., \alpha_{2^{b_s-1}-1}\}$ to obtain $\mathcal{T}^{msg}$:

$$\begin{aligned} &\mathcal{A}^{msg} \subset \mathcal{A}^c, T_1^{msg} = \alpha_1 L_1^{msg}, \\ &T_i^{msg} = \alpha_i L_{i-1}^{msg} + (1-\alpha_i) L_i^{msg}, \end{aligned} \tag{5}$$

where $2 \leq i \leq 2^{b_s-1}-1$. We remark that different techniques such as Non-Surjective FAID [17] and Information bottleneck [18] can be used to design Q^{msg}, but these alternative approaches are beyond the scope of this paper.

C. From QNN to FAID

From the discussion above, the proposed QNN has full precision of weights and biases, but low precision of activations and channel messages. When the offline training is done, we obtain trained weight matrices and biases $\{\mathbf{W}^{(3\ell)}, \mathbf{W}^{(3\ell+1)}, \mathbf{b}^{(3\ell)}, \mathbf{b}^{(3\ell+1)}\}$ for each ℓ-th iteration. Based on these weights and biases, we derive a series of message update rules. The rule of v_i's message update in the ℓ-th iteration is described as follows: $f_i^{(\ell)} : \mathcal{A}^c \times \{\mathcal{A}^{msg}\}^{d_{v_i}-1} \to \mathcal{A}^{msg}$,

$$f_i^{(\ell)}(y_Q, \mathbf{p}) = Q^{msg}(\Phi(\mathbf{E}[\mathbf{b}^{(3\ell+1)}] y_Q, \mathbf{w}_i^{(3\ell+1)} \mathbf{p})), \tag{6}$$

where $y_Q \in \mathcal{A}^c, \mathbf{p} \in \{\mathcal{A}^{msg}\}^{d_{v_i}-1}$, $\mathbf{w}_i^{(3\ell+1)}$ is the submatrix of $\mathbf{W}^{(3\ell+1)}$ associated with v_i, and $\mathbf{E}[\cdot]$ is the expectation operator. The hard decision rule $g_i^{(\ell)}$ can be obtained similar to $f_i^{(\ell)}$, except that the Q^{msg} in Eq. (6) is replaced by sign function, $\mathbf{w}_i^{(3\ell+1)}$ and $\mathbf{b}^{(3\ell+1)}$ are replaced by $\mathbf{w}_i^{(3\ell)}$ and $\mathbf{b}^{(3\ell)}$, and $\mathbf{p}$ in $g_i^{(\ell)}$ belongs to $\{\mathcal{A}^{msg}\}^{d_{v_i}}$. Then, the final FAID of a LDPC code is defined as $\mathcal{D}_{FAID} = (\mathcal{G}, \Psi, \{f_i^{(\ell)}\}, \{g_i^{(\ell)}\})$.

In principal, each variable node should have a message update rule in each iteration. For simplicity, consider regular LDPC codes, where all variable nodes have the same message update rule in each iteration. We force all nonzero entries in $\mathbf{W}^{(k)}$ to share the same value, i.e., $\mathbf{W}^{(k)}(i,j) = w^{(k)}$ if $\mathbf{W}^{(k)}(i,j)$ is a nonzero entry in $\mathbf{W}^{(k)}$. Then, Eq. (6) becomes

$$f^{(\ell)}(y_Q, \mathbf{p}) = Q^{msg}(\Phi(\mathbf{E}[\mathbf{b}^{(3\ell+1)}] y_Q, w^{(3\ell+1)} \mathbf{p})). \tag{7}$$

We remark that in this case, less parameters result in a faster learning, lower decoding complexity and lower memory requirements. When deploying $\mathcal{D}_{FAID}$ in hardwares, $\{f^{(\ell)}\}, \{g^{(\ell)}\}$ (or $\{f_i^{(\ell)}\}, \{g_i^{(\ell)}\}$) are implemented by integer-valued LUTs. As it is shown in [19], [20], a decoder supporting multiple decoding rules can be efficiently implemented in hardware, thus an iteration-dependent decoding rule requires only a small hardware overhead.

IV. Training with QNN

Since the channel is output-symmetric, and the QNN's activations preserve symmetry conditions, we can assume that the all-zero codeword is transmitted, i.e., $\mathbf{x} = \mathbf{0}$, thus $\mathbf{y} = \mathbf{1} + \mathbf{z}$. With the symmetry conditions on the weight matrices, it is sufficient to use a database composed of the realizations of the noisy vector $\mathbf{y}$. Let $\mathbf{r}^{(0)}$ 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 channel output vector $\mathbf{y}$.

A. BER-based objective function

The output $\mathbf{u}$ consists of the estimate of likelihood messages of a training sample. In most related works of using neural networks to optimize channel decoders, the binary cross entropy (BCE) function is widely applied as objective function, since it is a measurement of "soft" bit error rate, and it is differentiable everywhere. However, the BCE function is an approximation of BER. Training NNs to minimize BCE cannot guarantee to minimize BER. To see this, consider the following BCE function for each sample:

$$\Delta(\mathbf{u}, \mathbf{x}) = -\frac{1}{n} \sum_{i=1}^{n} x_i \log(1 - \sigma(u_i)) + (1 - x_i) \log(\sigma(u_i)),$$

where the $\sigma(\cdot)$ is the sigmoid function defined as $\sigma(x) = (1 + \mathrm{e}^{-x})^{-1}$. If the i-th bit is decoded correctly (equals to x_i) by the QNN, then there exists a positive value $0 < \epsilon_i < 1/2$ such that the i-th term in the summation of $\Delta(\mathbf{u}, \mathbf{x})$ equals to $\log(\frac{1}{2} + \epsilon_i)$; otherwise there exists a positive value $0 < \epsilon_i < 1/2$ such that the i-th term in the summation of $\Delta(\mathbf{u}, \mathbf{x})$ equals to $\log(\frac{1}{2} - \epsilon_i)$. Then $\Delta(\mathbf{u}, \mathbf{x})$ can be expressed as: $\Delta(\mathbf{u}, \mathbf{x}) =$

$\Delta_c + \Delta_e$, with $\Delta_c = \frac{1}{n}\sum_{j:x_j=\hat{x}_j} \log(1/2+\epsilon_j)^{-1}$ and $\Delta_e = \frac{1}{n}\sum_{j:x_j\neq\hat{x}_j} \log(1/2-\epsilon_j)^{-1}$, where $\hat{x}_j$ is the hard decision of u_j. Noted that Δ_c and Δ_e are the costs contributed by the bits decoded by QNN correctly and wrongly, respectively. Each term in the summation of Δ_c is within $[0,1]$, and each term in the summation of Δ_e is larger than 1.

Consider two samples $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, whose output values of QNN, denoted by $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$, satisfy the following conditions:

1) $\Delta(\mathbf{u}^{(1)},\mathbf{x}^{(1)}) = \Delta_c^{(1)} + \Delta_e^{(1)}, \Delta(\mathbf{u}^{(2)},\mathbf{x}^{(2)}) = \Delta_c^{(2)} + \Delta_e^{(2)}$;
2) There are d terms in $\Delta_e^{(1)}$ and $d-1$ terms in $\Delta_e^{(2)}$;
3) $|\Delta_c^{(1)} - \Delta_c^{(2)}| < \frac{1}{n}$; and
4) $d \max\limits_{j:x_j^{(1)}\neq\hat{x}_j^{(1)}} \log(1/2-\epsilon_j^{(1)})^{-1} + 1 \leq (d-1) \min\limits_{j:x_j^{(2)}\neq\hat{x}_j^{(2)}} \log(1/2-\epsilon_j^{(2)})^{-1}$.

Condition (2) implies that the first and second samples have d and $d-1$ bits in error after QNN decoding, respectively, and $\Delta_c^{(1)} < 1 - \frac{d}{n}, \Delta_c^{(2)} < 1 - \frac{d-1}{n}$. Condition (3) and (4) together indicate that the first sample has smaller BCE objective function, i.e., $\Delta(\mathbf{u}^{(1)},\mathbf{x}^{(1)}) < \Delta(\mathbf{u}^{(2)},\mathbf{x}^{(2)})$. In fact,

$$\begin{aligned}\Delta(\mathbf{u}^{(1)},\mathbf{x}^{(1)}) &= \Delta_c^{(1)} + \Delta_e^{(1)} < \Delta_c^{(2)} + \frac{1}{n} + \Delta_e^{(1)} \\ &\leq \Delta_c^{(2)} + \frac{1}{n} + \frac{d}{n} \max_{j:x_j\neq\hat{x}_j^{(1)}} \log(1/2-\epsilon_j^{(1)})^{-1} \\ &\leq \Delta_c^{(2)} + \frac{d-1}{n} \min_{j:x_j\neq\hat{x}_j^{(2)}} \log(1/2-\varepsilon_j^{(2)})^{-1} \\ &\leq \Delta_c^{(2)} + \Delta_e^{(2)} = \Delta(\mathbf{u}^{(2)},\mathbf{x}^{(2)}),\end{aligned}$$

hence a smaller BCE loss cannot guarantee a smaller BER.

In this work, instead of using BCE function, we consider the following bit error objective function for each sample, which measures the hamming distance between transmitted codeword $\mathbf{x}$ and the decoded codeword $\hat{\mathbf{x}}$, i.e.,

$$\Gamma(\mathbf{u},\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2, \text{ where } \hat{\mathbf{x}} = \frac{1-\mathrm{sgn}(\mathbf{u})}{2}. \quad (8)$$

$\Gamma(\mathbf{u},\mathbf{x})$ is an actual and practical metric to measure the performance of iterative decoders. In particular, according to Eq. (8), the bits which are decoded correctly neither make contributions to the objective function $\Gamma(\mathbf{u},\mathbf{x})$ nor the weight changes. On the other hand, the bits decoded wrongly are supposed to contribute to both $\Gamma(\mathbf{u},\mathbf{x})$ and gradients of weights and biases. However, $\Gamma(\mathbf{u},\mathbf{x})$ has derivatives of zero almost everywhere because of the sign function. To solve the zero gradients problem of Eq. (1) and (8), we apply straight-through estimators in the chain rule to calculate the gradients, which are introduced below.

B. Straight-through estimators

The straight-through estimator (STE) is a proxy derivative used in the QNN training to replace the zero derivative of quantization function in the chain rule [1]. Intuitively, STE is an estimate of the true partial gradient. In [1], it was found that the most efficient training of QNNs was using STEs, which was a good way to provide a non-trivial search direction.

To see how STE works for proposed QNN, consider the quantizers Q^{msg} in Eq. (1). In the backward propagation, by chain rule, the partial derivative of objective function $\Gamma(\mathbf{u},\mathbf{x})$ with respect to $\mathbf{W}^{(k)}$ is calculated as

$$\begin{aligned}\frac{\partial\Gamma(\mathbf{u},\mathbf{x})}{\partial\mathbf{W}^{(k)}} &= \frac{\partial\Gamma(\mathbf{u},\mathbf{x})}{\partial\mathbf{r}^{(k)}} \cdot \frac{\partial Q^{msg}}{\partial\mathbf{W}^{(k)}} \\ &= \frac{\partial\Gamma(\mathbf{u},\mathbf{x})}{\partial\mathbf{r}^{(k)}} \cdot \frac{\partial Q^{msg}}{\partial\Phi} \cdot \frac{\partial\Phi}{\partial\mathbf{W}^{(k)}}.\end{aligned} \quad (9)$$

Apparently, $\frac{\partial Q^{msg}}{\partial\Phi}$ is zero almost everywhere, making the weight updates still. To solve this, we use a proper surrogate derivative $\frac{\partial h}{\partial\Phi}$, called STE, to replace $\frac{\partial Q^{msg}}{\partial\Phi}$ in Eq. (9). Therefore, the partial derivative with respect to $\mathbf{W}^{(k)}$ can be approximated by

$$\gamma(\mathbf{W}^{(k)}) = \frac{\partial\Gamma(\mathbf{u},\mathbf{x})}{\partial\mathbf{r}^{(k)}} \cdot \frac{\partial h}{\partial\Phi} \cdot \frac{\partial\Phi}{\partial\mathbf{W}^{(k)}}. \quad (10)$$

As stated in [13], a STE should be designed carefully to guarantee that the negative expected proxy gradients are descent directions for minimizing the loss function. Namely, *the inner product between the expected proxy gradients and true gradients is nonnegative*:

$$\left\langle \mathbf{E}_{\mathbf{y}}[\gamma(\mathbf{W}^{(k)})], \frac{\partial\mathbf{E}_{\mathbf{y}}\left[\Gamma(\mathbf{u},\mathbf{x})\right]}{\partial\mathbf{W}^{(k)}} \right\rangle \geq 0. \quad (11)$$

Eq. (11) is called the *descent direction property*. How to design good STEs has been studied extensively in [10]–[13].

We introduce two STEs for Q^{msg} and the sign function used in Eq. (8). In particular, consider the following two functions:

$$h^{msg}(x) = \begin{cases} x & \text{if } |x| < T^{msg}_{2^{b_s-1}-1} \\ \mathrm{sgn}(x) L^{msg}_{2^{b_s-1}-1} & \text{otherwise} \end{cases}, \quad (12)$$

$$h^{\mathrm{sgn}}(x) = 2(1+e^{-x})^{-1} - 1, \quad (13)$$

whose gradients with respect to x are given below:

$$\frac{\partial h^{msg}(x)}{\partial x} = \begin{cases} 1 & \text{if } |x| < T^{msg}_{2^{b_s-1}-1} \\ 0 & \text{otherwise} \end{cases}, \quad (14)$$

$$\frac{\partial h^{\mathrm{sgn}}(x)}{\partial x} = \frac{2e^{-x}}{(1+e^{-x})^2}. \quad (15)$$

$\frac{\partial h^{msg}(x)}{\partial x}$ is used as the STE of Q^{msg}, and $\frac{\partial h^{\mathrm{sgn}}(x)}{\partial x}$ is used as the STE of sign function in Eq. (8). It is easy to see that $h^{msg}(x)$ and $h^{\mathrm{sgn}}(x)$ are approximations of Q^{msg} and sign function, respectively. Intuitively, their gradients are estimates of the true gradients of Q^{msg} and sign function. In particular, $\frac{\partial h^{msg}(x)}{\partial x}$ is called the clipped-ReLU STE, which has been shown to satisfy the above descent direction property [13] for minimizing the square sample loss of a two-linear-layer network, whose output is binary.

In principle, we can use clipped-ReLU STE for the sign function in Eq. (8) as well, which would provide a faster and simpler training. We postulate that the choice of STE is critically dependent on the optimality of Q^{msg}: if Q^{msg} is

well chosen, then a simple enough STE is good enough for training, whereas a complex STE may be necessary otherwise.

We use ADAM [21] with mini-batches for training, with gradients accumulated in full precision. The quantized activations and channel messages are used in both forward and backward propagation.

V. Numerical results

We built the QNN framework in Python3.6 and used Pytorch library for training. The training set consists of realizations of Gaussian noisy vectors. We constructed two QNNs 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 code, the same Q^{msg} and Q^c are used in both training and testing phases.

The simulation of $\mathcal{D}_{FAID}$ is carried out with the integer-valued LUTs of $\{f^{(\ell)}\}, \{g^{(\ell)}\}$, which are determined by the trained $\{w^{(3\ell)}, w^{(3\ell+1)}, \mathbf{b}^{(3\ell)}, \mathbf{b}^{(3\ell+1)}\}$, together with the quantizers Q^{msg} and Q^c. For comparisons with other conventional iterative decoders, we conduct simulations of MS decoding which are implemented with floating point and finite precisions. The measure of performance is *bit-error-rate* (BER) or *frame-error-rate* (FER). We compare the performance of $\mathcal{D}_{FAID}$ and MS decoding with same number of iterations.

A. Tanner code

The training set consists of 5000 samples at $E_b/N_0 = 4$dB. The number of epochs is 120, and the learning rate of ADAM is 0.01, with mini-batch size of 50. We maximize mutual information between channel input and the quantizer output to design a 4-bit quantizer Q^c for $E_b/N_0 = 6.5$dB. Hence $|\mathcal{A}^c| = 15$ and $|\mathcal{T}^c| = 7$. We take a symmetric subset of $\mathcal{A}^c$ of size 7 to form $\mathcal{A}^{msg}$, and select a set of scalars to decide $\mathcal{T}^{msg}$. To be specific, $L_1^{msg} = L_1^c, L_2^{msg} = L_4^c, L_3^{msg} = L_7^c$, and $\alpha_1 = \alpha_2 = \alpha_3 = 0.5$. As a result, Q^{msg} is of 3-bit and $|\mathcal{A}^{msg}| = 7$ and $|\mathcal{T}^{msg}| = 3$. We use $\frac{\partial h^{msg}(x)}{\partial x}$ and $\frac{\partial h^{\text{sgn}}(x)}{\partial x}$ in Eq. (14) and (15) as STEs of Q^{msg} and sign function, respectively. The maximum number of iterations is set to be 5. The MS with finite precision is quantized uniformly with 4-bit for both channel and message values, whose step is set to be 0.125.

In Fig. 2, we compare the BER and FER performance of $\mathcal{D}_{FAID}$, MS decoding with floating point, and MS decoding with 4-bit precision within 5 iterations. We also show the performance of floating-point offset MS (OMS) decoding as a reference (The offset is set to be 0.11). On average, with only 3-bit for message and 4-bit for channel output, $\mathcal{D}_{FAID}$ can achieve 0.2dB coding gain over MS decodings (both floating point and 4-bit precision) within 5 iterations.

B. Column-weight 4, medium-length code

For this QC-LDPC code (1296, 972), the training set consists of 5000 samples at $E_b/N_0 = 4$dB. The number of epochs is 110, and the learning rate of ADAM is 0.01, with mini-batch size of 10. We maximize mutual information between channel input and the quantizer output to design a 4-bit quantizer Q^c for $E_b/N_0 = 4$dB. Hence $|\mathcal{A}^c| = 15$ and $|\mathcal{T}^c| = 7$. We consider $\mathcal{A}^{msg} = \{0, \pm 0.2, \pm 0.5, \pm 1\}$, and $\mathcal{S}^{msg} = \{0.5, 0.5, 0.5\}$. Therefore, Q^{msg} is of 3-bit and $|\mathcal{A}^{msg}| = 7$ and $|\mathcal{T}^{msg}| = 3$. We use $\frac{\partial h^{msg}(x)}{\partial x}$ and $\frac{\partial h^{\text{sgn}}(x)}{\partial x}$ in Eq. (14) and (15) as STEs of Q^{msg} and sign function, respectively. The maximum number of iterations is set to be 5. The MS with finite precision is quantized uniformly with 4-bit for both channel and message values, whose step is set to be 0.1.

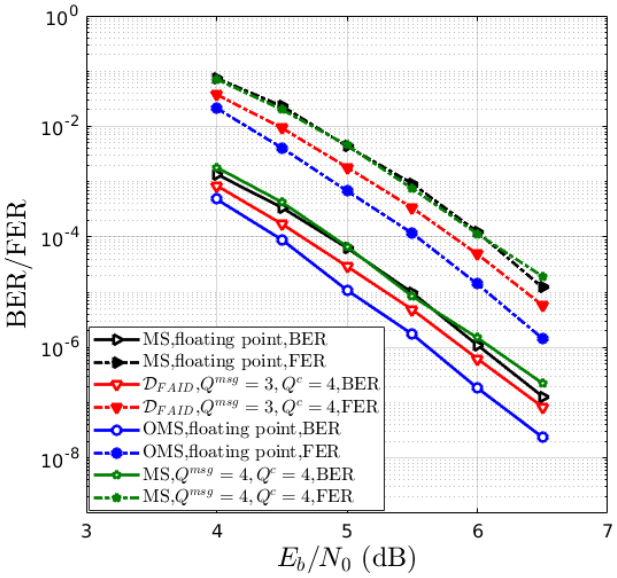

Fig. 2. BER and FER Performance of $\mathcal{D}_{FAID}$, MS and OMS decoding of Tanner code (155,64) within five iterations.

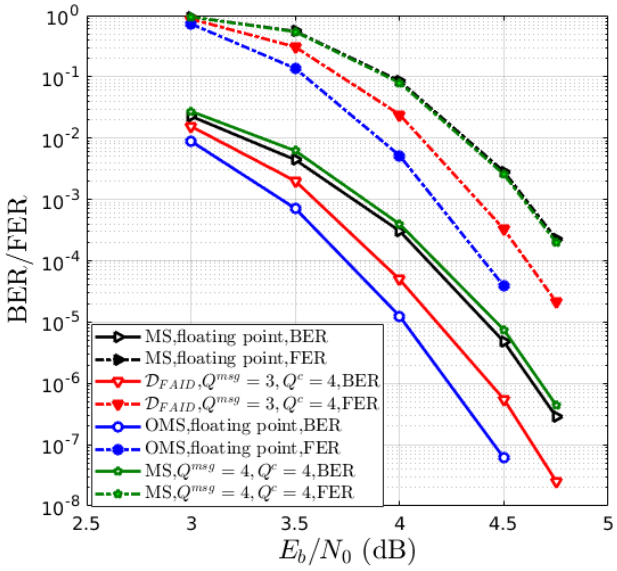

Fig. 3. BER and FER Performance of $\mathcal{D}_{FAID}$, MS and OMS decoding of QC-LDPC code (1296,972) within five iterations.

Fig. 3 gives the BER and FER performance of $\mathcal{D}_{FAID}$, MS decoding with floating point, and MS decoding with 4-bit precision within 5 iterations. The BER/FER performance of OMS decoding with floating point is also provided in Fig. 3 as reference (The offset is set to be 0.09). On average, with only 3-bit for message and 4-bit for channel output, $\mathcal{D}_{FAID}$ can achieve 0.2dB coding gain over both floating point MS decoding and 4-bit precision MS decoding within 5 iterations.

C. Related works on quantized message passing iterative decoders for LDPC codes

There have been a significant effort on designing FAIDs of LDPC codes for different channel models. For example, Planjery *at al.* design FAID message update rules to correct trapping sets and show that a FAID with only 3-bit precision outperforms the BP and all other message passing decoders over Binary Symmetric Channel (BSC) [22]. For AWGNC, Nguyen-Ly *at al.* [17] use density evolution to optimize the FAID. Meidlinger *at al.* [20] maximize the mutual information between decoding messages to build FAIDs, and demonstrate that a FAID with 3-bit of message and 4-bit of channel output can match MS decoding for regular LDPC codes. Lewandowsky *at al.* [18] apply Information bottleneck method to build FAIDs and demonstrate that a FAID with only 4-bit precision can approach to BP for finite length LDPC codes over AWGNC. In this work, we show the capability of a neural network on designing a FAID over AWGNC with competitive decoding performance. Compared with previous works, the QNN framework has more flexibility as there are plenty of choices on predefined quantizers, and it can be regarded as "end-to-end" design for a specific LDPC code.

VI. Conclusions and Discussions

In this paper, we explored the potential of QNNs to facilitate the design of finite-alphabet iterative decoding rules of LDPC codes. Predefined low precision quantizers were applied over activations and channel messages, with the quantizer of channel messages optimized by maximizing the mutual information. The low precision activations and channel messages were passed in both forward and backward propagation. In the training, we used the BER-based objective function, and introduced two STEs to around zero derivatives of Q^{msg} and the sign function used in hard decision. A FAID was derived based on the trained weights and biases, together with the predefined quantizers. Simulation results showed that within 5 iterations, the FAID with only 3-bit of message and 4-bit of channel output can achieve on average 0.2 dB over conventional floating point MS decoding and 4-bit precision MS decoding.

We remark that in this work, the σ for Q^c can be chosen to trade off the error floor and waterfall region. Open questions are left for future research, such as the joint quantizer optimization, and using QNNs for FAIDs over the BSC.

Acknowledgment

This work is funded by the NSF under grant NSF ECCS-1500170 and NSF SaTC-1813401.

References

[1] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[2] Y. Be'ery, "eforum on BP, LP and NN/ML decoding of HDPC codes," http://listserv.tau.ac.il/archives/hdpc-codes.html.

[3] O. Simeone, "A Very Brief Introduction to Machine Learning With Applications to Communication Systems," *ArXiv e-prints*, August 2018.

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

[5] E. Nachmani, E. Marciano, D. Burshtein, and Y. Beéry, "RNN decoding of linear block codes," 2017. [Online]. Available: https://arxiv.org/abs/1702.07560

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

[7] F. Liang, C. Shen, and F. Wu, "An iterative BP-CNN architecture for channel decoding," 2017. [Online]. Available: https://arxiv.org/abs/1707.05697

[8] 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.

[9] 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.

[10] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[11] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *arXiv preprint arXiv:1802.04680*, 2018.

[12] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin, "Blended coarse gradient descent for full quantization of deep neural networks," *arXiv preprint arXiv:1808.05240*, 2018.

[13] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Skh4jRcKQ

[14] W. Rave, "Quantization of log-likelihood ratios to maximize mutual information," *IEEE Signal Processing Letters*, vol. 16, no. 4, pp. 283–286, 2009.

[15] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[16] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, no. 1, pp. 7–12, 1960.

[17] 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. 26, no. 3, pp. 508–521, 2018.

[18] J. Lewandowsky and G. Bauch, "Information-optimum LDPC decoders based on the information bottleneck method," *IEEE Access*, vol. 6, pp. 4054–4071, 2018.

[19] F. Cai, X. Zhang, D. Declercq, B. Vasić, and S. K. Planjery, "Finite alphabet iterative decoders for LDPC codes: Optimization, architecture and analysis," *IEEE Transactions on Circuits and Systems - Part I: Regular Papers*, vol. 61, no. 5, pp. 1366–1375, May 2014.

[20] R. Ghanaatian, A. Balatsoukas-Stimming, T. C. Müller, M. Meidlinger, G. Matz, A. Teman, and A. Burg, "A 588-gb/s LDPC decoder based on finite-alphabet message passing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 2, pp. 329–340, 2018.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders-part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4033–4045, 2013.