Abstract—In this paper, we propose a new approach to design finite alphabet iterative decoders (FAIDs) for Low-Density Parity Check (LDPC) codes over binary symmetric channel (BSC) via recurrent quantized neural networks (RQNN). We focus on the linear FAID class and use RQNNs to optimize the message update look-up tables by jointly training their message levels and RQNN parameters. Existing neural networks for channel coding work well over Additive White Gaussian Noise Channel (AWGNC) but are inefficient over BSC due to the finite channel values of BSC fed into neural networks. We propose the bit error rate (BER) as the loss function to train the RQNNs over BSC. The low precision activations in the RQNN and quantization in the BER cause a critical issue that their gradients vanish almost everywhere, making it difficult to use classical backward propagation. We leverage straight-through estimators as surrogate gradients to tackle this issue and provide a joint training scheme. We show that the framework is flexible for various code lengths and column weights. Specifically, in high column weight case, it automatically designs low precision linear FAIDs with superior performance, lower complexity, and faster convergence than the floating-point belief propagation algorithms in waterfall region.

Index Terms—Binary symmetric channel, finite alphabet iterative decoders, low-density parity-check codes, quantized neural network, straight-through estimator.

I. INTRODUCTION

With the great potential of solving problems related to optimization, function approximation, inference etc., deep neural networks (DNN) have drawn intensive attention in communication, signal processing and channel coding communities in past three years. One popular way to use neural networks (NN) in these areas is to combine the model knowledge (or the prototype algorithms) and the NNs together, and use the optimization techniques of NNs to improve the model. Such NNs are called model-driven NNs, and deep unfolding [1] is shown to be a powerful example. As formalized by Hershey et al. [1], deep unfolding is to unfold the inference iterations as layers in a deep neural network, and a list of its applications to communication area can be found in [2]. In particular, for decoding error correction codes, deep unfolding has been shown to efficiently optimize various iterative decoding algorithms such as Belief Propagation (BP) [3]–[7]. One advantage of this approach is that the weight matrices and activation functions over hidden layers are constrained to preserve the message-passing symmetry conditions. This makes it possible to perform training on a single codeword and its noisy realizations, rather than on the entire code space, thus allowing to apply DNNs on long codes. Furthermore, in [8], it is shown that DNNs can speed the decoding convergence of conventional Min-Sum (MS) decoding algorithm. Unlike existing sophisticated theories like Density Evolution (DE) and Trapping Set (TS) used to guide decoder design for the waterfall and error-floor regions, respectively, there is no theory for improving decoding convergence speed. DNN decoders are promising candidates for the emerging ultra-reliable low-latency communication (URLLC) applications such as 5G, which need extra high throughput and low latency and can only allow a small number of iterations.

One obvious drawback of the above framework is that all the weight matrices used in DNN decoders are in floating-point precision, making the decoders prohibitively complex for most applications. To reduce the complexity of DNN decoders, recent works [9], [10] cluster weights into multiple groups after the floating-point weights are updated, and use a centroid value with finite precision to represent each group such that all the weights in the same group share the same value. Another shortcoming is that the current model-driven framework incorporates soft decision loss functions, which work well over Additive White Gaussian Noise Channel (AWGNC) where the values of messages, weights and biases are in continuous domain. However, it often encounters issues such as slow training convergence and poor generalization in terms of decoding performance when applied to discrete-output channels. The only work on model-driven neural networks for decoding over discrete-output channels is in [11] by Payani and Fekri, where the authors consider binary erasure channel (BEC) and recurrent neural networks (RNN), and propose differentiable Boolean logic algebra units in activation functions to learn the underlying logical flows.

Manuscript received September 17, 2019; revised February 13, 2020; accepted March 26, 2020. Date of publication April 6, 2020; date of current version July 15, 2020. The work of X. Xiao and B. Vasić was funded by the NSF under grant NSF SaTC-1813401 and NSF CCF-1855879. The work of V. Tandon was supported in part by the 2018 Keysight Early Career Professor Award, and by NSF under grants CAREER 1651492, and CNS 1715947. The associate editor coordinating the review of this article and approving it for publication was A. Graell i Amat. (Corresponding author: Xin Xiao.)

Xin Xiao, Bane Vasić, and Ravi Tandon are with the Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ 85721 USA (e-mail: 7xinxiao7@email.arizona.edu; vasic@ece.arizona.edu; tandonr@email.arizona.edu). Shu Lin is with the Department of Electrical and Computer Engineering, University of California Davis, Davis, CA 95616 USA (e-mail: shulin@ucdavis.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCOMM.2020.2985678
Unlike other papers, in this work we consider binary symmetric channel (BSC), and use model-driven neural networks to design decoders with finite precision for Low-Density Parity Check (LDPC) codes. Moreover, our goal is to design such decoders with fast convergence. Decoders with finite precision, referred to finite alphabet iterative decoders (FAIDs) [12], are not only of great theoretical value, but also practical importance. The general FAID framework offers great flexibility in designing iterative decoders for LDPC codes and results in decoders with excellent waterfall and error floor properties. Such decoders are appealing to URLLC applications as they can provide high throughput and less decoding complexity. To make the model-driven NNs work over BSC, we use bit error rate (BER) as a loss function, which is not only a hard decision loss function, but a more common and accurate metric to measure the decoding performance of iterative decoders as well. NNs with finite precision can be modeled as Quantized NNs (QNN). However, training QNNs with BER loss function directly in general encounters a critical issue caused by the low precision activation functions in the QNN and the quantization of the BER, that their gradients vanish almost everywhere, making it difficult to use classical backward propagation. Straight-through estimators (STE) [13] are surrogate derivatives used in the backward propagation to handle the zero gradient issue (See Section IV. B). In our prior work [14], we show that by replacing zero derivatives of the quantizations with STEs in the chain rule, we are able to train model-driven QNNs and design FAIDs with 3-bit messages and 4-bit channel outputs that outperform floating-point MS decoder over AWGNC.

A. Main Contributions of this Paper

In this paper, we extend the QNN framework in [14] and apply it to BSC. Our main contributions can be summarized as follows: 1) We propose a recurrent QNN (RQNN)-aided approach to design linear FAIDs for LDPC codes, to the best of the knowledge, first model-driven NN decoder designed over BSC. In brief, we use RQNNs to optimize linear FAIDs for regular LDPC codes, whose variable node update function takes as input a linear combination of incoming messages and channel value. The recurrent structure shares the trainable parameters across all iterations, which results in fewer weights and biases than the forward multi-layer DNNs and makes it possible to generalize to more iterations. 2) We propose BER loss function and provide a training scheme to train the RQNN. Instead of predefining the quantizers for messages as in [14], we jointly train the message levels and the parameters in the RQNN (including channel value, weights, and biases) to enlarge the searching space, thus increasing the possibility of finding good local minima. The basic idea is to simultaneously compute the sample proxy gradients of levels and parameters in the RQNN, which are obtained via the chain rule revised by STEs. 3) We show that this framework is quite flexible by conducting simulations on various LDPC codes with different column weights and code lengths. The examples and numerical results validate that the RQNN-aided linear FAIDs with 3-bit precision perform as well as floating-point BP algorithms. In particular, for column-weight-four Quasi-Cyclic (QC) LDPC codes, they are capable of surpassing the floating-point BP algorithms, with much lower complexity and faster convergence. This makes it possible to design low precision linear FAIDs with superior performance for high column weight scenarios, where the existing design methods are not applicable due to high complexity.

II. PRELIMINARIES

A. Notation

In this paper, we consider a binary LDPC code \( C \), with associated parity check matrix \( H \) of size \( M \times N \). Let \( G = (V, C, E) \) be its Tanner graph, where \( V \) (respectively, \( C \)) is the set of \( N \) variable (respectively, \( M \) check) nodes, and \( E \) is the set of edges with \( I = |E| \). We 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 \( (e) \), \( 1 \leq i \leq N, 1 \leq j \leq M, 1 \leq e \leq I \). The degree of a node in \( G \) is defined as the number of its neighbors in \( G \). If all variable nodes in \( V \) have the same degree \( d_v \), we say that \( C \) has regular column weight \( d_v \); and if all check nodes in \( C \) have the same degree \( d_c \), we say that \( C \) has regular row weight \( d_c \). Let \( x = (x_1, x_2, ... x_N) \in C \) be the transmitted codeword, and \( y = (y_1, y_2, ... y_N) \) be the received channel output vector. Denote the crossover probability of BSC by \( p \).

B. Definition of Linear FAID

Next we introduce the definition of a FAID [12]. A \( b \)-bit FAID denoted by \( D_{FAID} \) can be defined by a 5-tuple:

\[
D_{FAID} = (M, \mathcal{Y}, \Phi, \Psi, \Upsilon)
\]

\( M \) is the domain of the messages passing in FAID, and is defined as \( M = \{0, \pm L_1, \pm L_2, ..., \pm L_s\} \), with \( L_i \in \mathbb{R}^+ \), \( 1 \leq i \leq s, s \leq 2^{b-1} - 1 \) and \( L_i > L_j \) if \( i > j \). For a message \( m \in M \) associated with \( v_i \), its sign represents an estimate of the bit of \( v_i \), namely \( v_i = 0 \) if \( m > 0 \), \( v_i = 1 \) if \( m < 0 \), and \( v_i = y_i \) if \( m = 0 \), and its magnitude \( |m| \) measures the reliability of this estimate. \( \mathcal{Y} \) is the domain of channel outputs. For BSC, \( \mathcal{Y} = \{\pm C\} \) with some \( C \in \mathbb{R}^+ \) as we use the bipolar mapping: \( 0 \to -C \) and \( 1 \to +C \). Let \( z = (z_1, z_2, ... z_N) \) be the input vector to a FAID, with \( z_i = (-1)^{y_i} C, 1 \leq i \leq N \). The functions \( \Phi \) and \( \Psi \) describe the message update rules of variable nodes and check nodes, respectively. For a check node \( c_j \) with degree \( d_c \), its updating rule is given by

\[
\Psi(m_j) = \prod_{m \in m_j} \text{sgn}(m) \cdot \min_{m \in m_j} |m|
\]

where \( \text{sgn} \) is the sign function and \( m_j \) is the set of extrinsic incoming messages to \( c_j \), with \( |m_j| = d_c - 1 \) and \( m_j \in \mathcal{M}^{d_c-1} \). For a variable node \( v_i \) with degree \( d_v \), its updating rule is given by

\[
\Phi(z_i, n_i) = Q\left( \sum_{m \in n_i} m + \omega_i z_i \right)
\]

where \( n_i \) is the set of extrinsic incoming messages to \( v_i \), and \( z_i \) is the bipolar mapping of \( v_i \), with \( |n_i| = d_v - 1 \) and \( n_i \in \mathcal{M}^{d_v-1} \). The function \( Q(\cdot) \) is the quantizer defined by \( M \).
and a threshold set \( T = \{T_1, ..., T_s, T_{s+1} = \infty\} \), with \( T_i \in \mathbb{R}^+ \), \( 1 \leq i \leq s \) and \( T_i > T_j \) for any \( i > j \):

\[
Q(x) = \begin{cases} 
\text{sgn}(x)L_i & \text{if } T_i \leq |x| < T_{i+1} \\
0 & \text{if } |x| < T_1 
\end{cases}
\]  

(1)

The coefficient \( \omega_i \) is a non-negative real number, which is computed using a symmetric function of \( \ell_i \). If \( \omega_i \) is a constant for all possible \( \ell_i \), \( \Phi \) is the quantization of a linear function, and its associated FAID is called linear FAID.

At the end of each iteration, the estimate of bit associated with each variable node \( v_i \) is made by the sign of the sum of all incoming messages and channel value \( z_i \), i.e., zero if the sum is positive, one if the sum is negative, and \( y_i \) if the sum is zero. This sum represents the estimate of bit-likelihoods and we denote this operation by \( \Upsilon \).

From the definitions, \( M \) is the codomain of both \( \Phi \) and \( \Psi \). Note that the linear FAIDs have the same check node updating rules \( \Psi \) and bit-likelihood approximator \( \Upsilon \) as MS decoder, thus a linear FAID is completely determined by \( \Phi \). In the following Section, we discuss how to use an RQNN to design a linear FAID via optimizing \( \Phi \) and \( \Upsilon \).

III. AN RQNN FRAMEWORK FOR LINEAR FAID

A. A Model-Driven Structure for Linear FAID

The proposed RQNN is a model-driven recurrent deep neural network, which is constructed by unfolding the linear FAID with a given number of iterations, as shown in [5]. The connection between consecutive layers is determined by the Tanner graph \( G \), and activation functions over hidden layers are defined based on \( \Phi, \Psi \), and \( \Upsilon \). As it has recurrent structure, the trainable parameters are shared among all the iterations. More specifically, suppose that the RQNN consists of \( K \) hidden layers, with output values denoted by \( r^{(k)}, 1 \leq k \leq K \). \( r^{(0)} \) (respectively, \( r^{(K+1)} \)) represents the values of input (respectively, output) layer. \( r^{(k)} = (r_1^{(k)}, r_2^{(k)}, ..., r_{J_k}^{(k)})^{T} \), 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 in \( k \)-th layer, \( 0 \leq k \leq K+1, 1 \leq t \leq J_k \). As shown in Fig. 1, three consecutive hidden layers in one column correspond to one iteration in linear FAID. For \( \ell \geq 0 \), \( r^{(3\ell+1)} \) (respectively, \( r^{(3\ell+2)} \)) represents the variable (respectively, check) nodes message update, and \( r^{(3\ell)} \) represents the estimate of bit-likelihoods except for \( r^{(0)} \). Each neuron stands for an edge in variable and check nodes message update layers. Each neuron in bit-likelihood approximation layers stands for a variable node. Therefore, we have \( J_k = N \) if \( 3|k \); otherwise \( J_k = I \). Because of the recurrent structure, there are three weight matrices \( W_0 \) (of size \( I \times N \)), \( W_1 \) (of size \( I \times I \)) and \( W_2 \) (of size \( N \times I \)) used in initialization, variable node update, and bit-likelihood approximation, respectively. Similarly, there are two bias vectors \( b_1 \) and \( b_2 \) used in variable node update and bit-likelihood approximation, respectively. The \((i,j)\)-th entry of \( W_k \) is denoted as \( W_k(i,j) \), and the \((i)\)-th entry of \( b_k \) is denoted as \( b_k(i) \).

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

\[
r^{(k)} = \begin{cases} 
\Phi(b_1, z^{T}, W_1 r^{(k-2)}), & 3(k-1), \\
\Psi(r^{(k-1)}), & 3(k-2), \\
\Upsilon(b_2, z^{'}, W_2 r^{(k-1)}), & 3k, 
\end{cases}
\]  

(2)

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

\[
r^{(e)}_v = Q\left( \sum_{(e'=v_i,c_j),t \neq j} W_1(e,e')r^{(k-2)} + b_1(e)v_i \right),
\]

if \( 3|k-2 \),

\[
r^{(e)}_c = \prod_{(e'=v_i,c_j),t \neq i} \text{sgn}(r^{(k-1)}_{e'}), \min_{(e'=v_i,c_j),t \neq i} \left| r^{(k-1)}_{e'} \right|,
\]

and if \( 3|k,\)

\[
r^{(k)}_v = \sum_{(e'=v_i,c_j)} W_2(i,e')r^{(k-1)} + b_2(i)v_i.
\]

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

\[
r^{(1)} = Q(W_0 r^{(0)}),
\]

(3)

where \( r^{(0)} = z^{T} = (-1)^T \cdot C, r^{(1)} = Q(W_0(v_i,i)r^{(0)}), \forall (e) = (v_i, c_j) \). At each \( k \)-th layer where \( 3|k \) (i.e., the bit-likelihood estimate layer), for each training sample, we check whether its estimated codeword satisfies all parity equations or not. If so, this training sample will skip the remaining subsequent layers and its output at the current \( k \)-th layer will be directly used to calculate the loss function, namely, \( r^{(K+1)} = r^{(k)} \), if \( \frac{1}{2} \left( 1 - \text{sgn}(r^{(k)})^{T} \right) \cdot H^{T} = 0 \). If there is no intermediate bit-likelihood estimate layer satisfying all parity equations, the last bit-likelihood estimate layer will be used to compute the loss function. Furthermore, as we consider regular LDPC codes, the trained model will have the same decoding performance no matter whether we put the constraint during training or not, which forces all the edges to share the same weight. Hence all nonzero entries in \( W_k \) are constrained to have the same value, i.e., \( W_k(i,j) = w_k \) if \( W_k(i,j) \) is a nonzero entry in \( W_k \), and \( W_k \) is equivalent to the product of a trainable single scalar \( w_k \) and a constant connection matrix. Denote the
constant connection matrix in initialization, variable node update layer and bit-likelihood estimate layer by $B_0$, $B_1$ and $B_2$, respectively. $B_0$, $B_1$ and $B_2$ can be determined in the following way: $B_0$ has size of $I \times N$, and for the $(i,j)$-th entry in $B_0$, $1 \leq i \leq I$, $1 \leq j \leq N$, $B_0(i,j) = 1$ if the edge $(i)$ is incident to $v_j$, otherwise $B_0(i,j) = 0$; $B_1$ has size of $I \times I$, and for the $(i,j)$-th entry in $B_1$, $1 \leq i \leq I$, $1 < j < I$, $B_1(i,j) = 1$ if $i \neq j$ and both edge $(i)$ and edge $(j)$ are incident to the same variable node, otherwise $B_1(i,j) = 0$; $B_2$ has size of $N \times I$, and $B_2 = B_2^T$.

The $Q(z)$ function in Eq. (2) and (3) is defined as in Section II. The channel value $C$ and the levels $L_i$ in $M$ are also parameters to be learned during the training process. The threshold set is constrained by $M$ such that $T_i$ is within the range $[L_{i-1}, L_i]$, $i \geq 2$. To satisfy this constraint, we use a set of scalars $S = \{\alpha_1, \alpha_2, ..., \alpha_s\}$ to obtain $T$ by

$$T_1 = \alpha_1 L_1, T_i = \alpha_i L_{i-1} + (1 - \alpha_i) L_i, i \geq 2,$$

(4) where $\alpha_i \in [0, 1], 2 \leq i \leq s, 0 < \alpha_1 \leq 1$. $S$ determines the relationship between $M$ and $T$. In our framework, we redefine $S$ so that the trainable parameters would adjust themselves to the relationship constrained by $S$.

### B. From an RQNN to a Linear FAID

The RQNN proposed above has weights and biases of full precision and a finite number of activation outputs (messages) of full precision. The trainable parameters include the finite alphabet set $M$, channel value $C$, weights $\{w_0, w_1, w_2\}$ and biases $\{b_1, b_2\}$. We denote the set of trainable parameters by $\Theta$, i.e., $\Theta \triangleq \{M, C, w_0, w_1, w_2, b_1, b_2\}$. When the offline training is completed, we obtain a trained $\Theta$, based on which we derive two look-up tables (LUTs), which are used to describe $\Phi$ and the hard decision $sgn(T)$, and can be efficiently implemented in hardware. In the initialization, the channel value $+C$ (respectively, $-C$) is first quantized by $+Q(w_0 C)$ (respectively, $-Q(w_0 C)$). For the case that the received channel value is $-C$, the LUT used to describe $\Phi$ for all variable nodes is a $(d_v - 1)$-dimensional array, denoted as $\Phi_{RQNN}$. Let $M = \{M_1, M_2, ..., M_{2^{d_v} + 1}\}$, with $M_1 = -L_s$, $M_2 = -L_{s-1}, ..., M_s = -L_1, M_{s+1} = 0, M_{s+2} = L_1, M_{s+3} = L_2, ..., M_{2s+1} = L_s$. Then, for $1 \leq i_1, i_2, ..., i_{d_v} - 1 \leq 2s + 1$, the entry $(i_1, i_2, ..., i_{d_v} - 1)$ of $\Phi_{RQNN}$ is computed by

$$\Phi_{RQNN}(i_1, i_2, ..., i_{d_v} - 1) = Q\left(-\overline{b_1} \cdot C + w_1 \sum_{j=1}^{d_v - 1} M_{i_j}\right),$$

(5) where $\overline{b_1}$ is the average of the bias vector $b_1$ which is computed as $\overline{b_1} \triangleq \frac{1}{I} \sum_{j=1}^{I} b_1(j)$. We remark that the distribution of trained biases is almost centered at the mean, and it makes no difference regarding the decoding performance whether we use $\overline{b_1}$ or $b_1(j)$ for each individual edge. Therefore we can simply use $\overline{b_1}$ in Eq. (5) so that we just need one LUT. We also point out that it is helpful to give more freedom to the bias vector during training, namely, it is better to set the biases in the same layer as independent, rather than binding them together. Since $\Phi_{RQNN}$ is symmetric, the LUT of the case that the received channel value is $C$ can be obtained simply with $-\Phi_{RQNN}(2s + 2 - i_1, 2s + 2 - i_2, ..., 2s + 2 - i_{d_v} - 1)$. For column-weight-three LDPC codes, $\Phi_{RQNN}$ is a 2-dimensional array, and is a 3-dimensional array for column-weight-four LDPC codes.

Similarly, the LUT used to describe the hard decision $sgn(T_d)$ for all variable nodes is a $d_v$-dimensional array, denoted as $\Upsilon_d$, for the case that the received channel value is $-C$. For $1 \leq i_1, i_2, ..., i_{d_v} \leq 2s + 1$, the entry $(i_1, i_2, ..., i_{d_v})$ of $\Upsilon_d$ is computed by

$$\Upsilon_d(i_1, i_2, ..., i_{d_v}) = \text{sgn}\left(-\overline{b_2} \cdot C + w_2 \sum_{j=1}^{d_v} M_{i_j}\right),$$

where $\overline{b_2}$ is the average of the bias vector $b_2$ which is computed as $\overline{b_2} \triangleq \frac{1}{N} \sum_{j=1}^{N} b_2(j)$. Since $\Upsilon_d$ is also symmetric, the LUT of the case that the received channel value is $C$ is computed as $-\Upsilon_d(2s + 2 - i_1, 2s + 2 - i_2, ..., 2s + 2 - i_{d_v})$. Because of the recurrent structure, such RQNN corresponds to a fixed linear FAID with 8-bit precision, which is defined as $\mathcal{D}_{FAID} = \{M, \{\pm C\}, \Phi_{RQNN}, \Upsilon_d\}$. In the next Section, we will discuss how to jointly train the RQNN, together with the finite alphabet set $M$ and the channel value $C$.

### IV. Training RQNNs

Let the RQNN corresponding to $\mathcal{D}_{RQNN}$ be denoted as $\mathcal{D}_{RQNN}$. Similar to the conventional iterative decoders, the weights and biases in the RQNN are initialized to be 1. This implies that the training process starts from a quantized min-sum decoder. As mentioned in [15], the initialization points of deep learning models affect the convergence of the training process, and are usually designed heuristically. In our framework, the initial linear FAID, defined by the initial values of $M, C$ and predefined $S$, needs to satisfy the Property II in [12] to ensure a good initialization point. We rephrase this property as follows.

**Property 1:** Monotonicity of initial linear FAID.

Let $z \in \mathcal{Y}, \{m_1, ..., m_{d_v - 1}\}, \{m_1', ..., m_{d_v - 1}'\} \in \mathcal{M}^{d_v - 1}$, and $Q$ be the quantizer defined in Eq. (1). Then

$$Q\left(\sum_{j=1}^{d_v - 1} m_j + z\right) \geq Q\left(\sum_{j=1}^{d_v - 1} m_j' + z\right),$$

when $m_j \geq m_j', \forall j \in \{1, ..., d_v - 1\}$. Property 1 is a typical property in existing iterative decoders, which guarantees that for a given channel value, the output does not decrease as the values of the incoming messages increase.

Since the channel is output-symmetric, and the RQNN’s activations preserve symmetry conditions, we can assume that the all-zero codeword is transmitted, i.e., $x = 0$. With the symmetry conditions on the weight matrices, it is sufficient to use a database composed of the realizations of the noisy bipolar vector $z$. Let $\mathbf{r}^{(0)}$ and $\mathbf{u} = \mathbf{r}^{(K+1)}$ be the values in the input and output layer, respectively, i.e., $\mathbf{u} = \mathcal{D}_{RQNN}(\mathbf{r}^{(0)})$. 

---

*IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. 68, NO. 7, JULY 2020*
Both $\mathbf{r}^{(0)}$ and $\mathbf{u}$ have length $J_0 = J_{K+1} = N$, with $\mathbf{r}^{(0)}$ receiving the channel bipolar vector $\mathbf{z}^T$.

A. BER-Based Loss Function

The output $\mathbf{u}$ is composed of the estimates of bit-likelihoods of each training sample, which are real numbers. In most related work of using neural networks to optimize channel decoders, soft decision loss functions (the binary cross entropy function for instance), are widely employed (as in [3]–[5], [7]–[10], [16]) since they measure “soft” bit error rate and are differentiable everywhere. They work well over AWGN channels where the values of messages, weights and biases are in continuous domain. However, when applied to discrete-output channels such as BEC, it often encounters slow convergence and poor generalization issues [11]. We also face the same issues when deploying the model-driven DNNs with the soft decision loss functions over BSC. Furthermore, the soft decision loss functions are approximations of BER. Training NNs to minimize soft decision loss functions cannot guarantee minimization of BER. In this work, instead of soft decision loss functions, we consider the following bit error rate loss function for each sample, which measures the Hamming distance between transmitted codeword $\mathbf{x}$ and the decoded codeword $\hat{\mathbf{x}}$, i.e.,

$$\Gamma(\mathbf{u}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2,$$

where $\hat{x} = \frac{1}{2} \left( -sgn(u^T) \right)$. $\Gamma(\mathbf{u}, \mathbf{x})$ is a hard decision loss function, and it is a widely used and practical metric to measure the decoding performance of iterative decoders. Therefore, the learning task of the RQNN is to minimize the expected loss given a finite number of samples, i.e., $\min_{\Theta} \mathbb{E}_y [\Gamma(\mathbf{u}, \mathbf{x})]$, where $\mathbb{E}_y [\cdot]$ is the expectation operator over random vector $\mathbf{y}$. During the training, the partial gradients of the expected loss with respect to the trainable parameters in $\Theta$, \{$\frac{\partial \mathbb{E}_y [\Gamma(\mathbf{u}, \mathbf{x})]}{\partial \theta} : \theta \in \Theta$\}, are not accessible. Instead, we can only compute the empirical sample partial gradients given a finite number of samples, which become the expected sample partial gradients when the number of samples goes to infinity, i.e., $\{ \mathbb{E}_y [\Gamma(\mathbf{u}, \mathbf{x})] : \theta \in \Theta \}$. In the RQNN framework, $\frac{\partial \mathbb{E}_y [\Gamma(\mathbf{u}, \mathbf{x})]}{\partial \theta}$ and $\mathbb{E}_y [\frac{\partial \Gamma(\mathbf{u}, \mathbf{x})}{\partial \theta}]$ are not the same, because the loss function $\Gamma(\mathbf{u}, \mathbf{x})$ in (6) and the Quantizer $Q(\cdot)$ in Eq. (2) have derivatives of zero almost everywhere, making each sample partial gradient zero and thus $\mathbb{E}_y [\frac{\partial \Gamma(\mathbf{u}, \mathbf{x})}{\partial \theta}] = 0$. To solve the zero-gradient problems of Eq. (2) and (6), we apply straight-through estimators in the chain rule, which are introduced next.

B. Straight-Through Estimators

The straight-through estimator (STE) is a surrogate derivative used in the QNN training to replace the zero derivative of a quantization function in the chain rule [13]. Intuitively, for $\theta \in \Theta$, the amount computed via the chain rule which is revised with STE (s) is an artificial gradient, and is used to update $\theta$. The parameter changes made by these artificial gradients need to form a descending direction to minimize the expected loss, which is approximately the same as the descending direction formed by the negations of the partial gradients of the expected loss. In [13], it is found that the most efficient training of QNNs is using STEs, which is a good way to provide a non-trivial search direction.

To see how STEs work for our RQNN, we take a look at the backward propagation. Given a sample $⟨\mathbf{y}, \mathbf{x}⟩$, the sample partial derivative of the loss function $Γ(\mathbf{u}, \mathbf{x})$ (as defined in Eq. (6)) with respect to $w_k$ is calculated as follows

$$\frac{\partial Γ(\mathbf{u}, \mathbf{x})}{\partial w_k} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i) \frac{\partial (sgn(u_i))}{\partial u_i} \frac{\partial u_i}{\partial w_k},$$

where $\frac{\partial u_i}{\partial w_k}$ can be further expanded via the chain rule. For $k = 0, 1$, it is expected that there exist several terms in the form of $\frac{\partial q(r)}{\partial r}$ in the expanded expression of $\frac{\partial u_i}{\partial w_k}$ because of the composition of layers. For $k = 2$, $\frac{\partial u_i}{\partial w_k}$ is nothing but the sum of all incoming messages to $v_i$. These $\frac{\partial q(r)}{\partial r}$s and $\frac{\partial u_i}{\partial w_k}$ are zero almost everywhere, making the weight updates stagnant. We use two proper surrogate derivatives $\frac{\partial h_{\text{sym}}(u_i)}{\partial u_i}$ and $\frac{\partial h_{\text{M}}(r)}{\partial r}$, called STE, to replace $\frac{\partial q(r)}{\partial r}$ in Eq. (7) and every $\frac{\partial q(r)}{\partial r}$ (if any) in the expanded expression of $\frac{\partial u_i}{\partial w_k}$, respectively. Then the sample proxy gradient $γ(w_k)$ that we use to update $w_k$ is computed as

$$γ(w_k) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i) \frac{\partial h_{\text{sgn}}(u_i)}{\partial u_i} \frac{\partial u_i}{\partial w_k}.$$

In Eq. (8), $\frac{\partial h_{\text{sym}}(u_i)}{\partial u_i}$ is the same as $\frac{\partial u_i}{\partial w_k}$ if there is no term in the form of $\frac{\partial q(r)}{\partial r}$ in the expanded expression of $\frac{\partial u_i}{\partial w_k}$. Otherwise, $\frac{\partial u_i}{\partial w_k}$ is obtained by the expanded expression of $\frac{\partial u_i}{\partial w_k}$ modified with $\frac{\partial h_{\text{M}}(r)}{\partial r}$. We can define the sample proxy gradients for other parameters in $\Theta$ in the same way. Intuitively, an STE should be designed carefully to guarantee that the parameter changes are in descent directions in order to minimize the expected loss. To get a better understanding of the behavior of STEs, we consider a simple scenario, where for each parameter $θ \in \Theta$, we use the following rule to update $θ$ at time $t + 1$:

$$θ^{t+1} ← θ^t - η \mathbb{E}_y [γ(θ^t)].$$

In Eq. (9), $η$ is the learning rate or stepsize with $0 < η < 1$. We further view $\Theta$ as a row vector and use $f(\Theta)$ to represent the expected loss of the model, which takes $\Theta$ as parameters. By the Taylor expansion,

$$f (\Theta^{t+1}) - f (\Theta^t) = (\Theta^{t+1} - \Theta^t) \nabla f (\Theta^t)^T + \frac{1}{2!} (\Theta^{t+1} - \Theta^t)^T H_f (\Theta^t) (\Theta^{t+1} - \Theta^t)^T + ..., \quad (10)$$

where $H_f (\Theta^t)$ is the Hessian matrix of the expected loss evaluated at $θ^t$. According to Eq. (9), $θ^{t+1} - θ^t = -η \mathbb{E}_y [γ(θ^t)].$ When we take a small $η$, Eq. (10) can be approximated by the first term, namely,

$$f (\Theta^{t+1}) - f (\Theta^t) ≈ -η \mathbb{E}_y [γ(θ^t)] \nabla f (\Theta^t)^T. \quad (11)$$
It is concluded that if the right hand side of Eq. (11) is negative, the training will decrease the expected loss in time $t + 1$. Furthermore, $E_y[\gamma(\Theta^t)]\nabla f(\Theta^T)$ is nothing but the inner product between $E_y[\gamma(\Theta^t)]$ and $\nabla f(\Theta^t)$. We summarize this descent direction property for minimizing the expected loss as follows.

**Property 2: Descent direction property.**

Let the model take $\Theta$ as parameters and follow Eq. (9) to update $\Theta$. Then the inner product between the expected proxy gradients $E_y[\gamma(\Theta)]$ and partial gradients of the expected loss is nonnegative, i.e.,

$$\langle E_y[\gamma(\Theta)], \nabla E_y[\Gamma(u, x)] \rangle \geq 0.$$  
(12)

This implies that the vector of parameter changes $-\eta E_y[\gamma(\Theta)]$ is roughly in the same direction as $-\eta \nabla E_y[\Gamma(u, x)]$. In general, the first term in Eq. (12) indicates how the training moves in each parameter dimension, and $E_y[\gamma(\Theta)]$ can be replaced by other changes made by more sophisticated rules (for instance, ADAM [17]) for training speed and stability considerations. How to design good STEs has been studied extensively in [18]–[22].

We introduce two STEs for $Q$ and the sign function used in Eq. (6) with respect to the parameters in $\Theta$. To be specific, consider the following two functions

$$h^M(x) = \begin{cases} x & \text{if } |x| < T_s, \\ \text{sgn}(x) L_s & \text{otherwise}, \end{cases}$$  
$$h^{\text{sgn}}(x) = 2(1 + e^{-x})^{-1} - 1.$$  
(13)

$h^M(x)$ is a variant of the hard tanh function, and $h^{\text{sgn}}(x)$ is $\tanh(x/2)$. Intuitively, $h^M(x)$ and $h^{\text{sgn}}(x)$ are approximations of $Q$ and sign function, respectively (as shown in Fig. 2(a) and (b)). Their gradients with respect to $x$ are given below:

$$\frac{\partial h^M(x)}{\partial x} = \begin{cases} 1 & \text{if } |x| < T_s, \\ 0 & \text{otherwise}, \end{cases}$$  
$$\frac{\partial h^{\text{sgn}}(x)}{\partial x} = \frac{2e^{-x}}{(1 + e^{-x})^2}.$$  
(14)
(15)

In the RQNN framework, $\frac{\partial h^M(x)}{\partial x}$ is used as the STE of $Q$, and $\frac{\partial h^{\text{sgn}}(x)}{\partial x}$ is used as the STE of the sign function in Eq. (6). In particular, $\frac{\partial h^M(x)}{\partial x}$ is known as the saturated STE [22] for the case $T_s = 1$, which has been shown to be equivalent to stochastic binarization proposed in [18]. Bengio et al. [13] use numerical results to show that $\frac{\partial h^{\text{sgn}}(x)}{\partial x}$ provides the best validation and test error among several types of solutions to the zero-gradient issues in the QNN. In principle, we can use the saturated STE for the sign function in Eq. (6) as well, which would provide a faster and simpler training. However, the STE $\frac{\partial h^{\text{sgn}}(x)}{\partial x}$ in Eq. (15) leads to a more stable training process than the saturated STE in our RQNN framework. Therefore we make a trade-off between training performance and complexity by choosing the saturated STE for $Q$ and $\frac{\partial h^{\text{sgn}}(x)}{\partial x}$ for sign function.

### C. Joint Training

In this subsection, we introduce a joint training scheme, where the “joint” indicates that both the levels in the alphabet set $\mathcal{M}$ and the RQNN parameters (channel value $C$, weights and biases) are trained simultaneously.

To make the alphabet set $\mathcal{M}$ trainable and satisfy the constraints $L_i > L_j$ if $i > j$, we define $\mathcal{M}$ in the following way:

$$L_1 = 1, L_k = 1\prod_{i=1}^{k-1} l_i, 2 \leq k \leq s,$$  
(16)

where $l_i, 1 \leq i \leq s - 1$ are to be learned with constraints $l_i > 1, \forall i \geq 1$, and are initialized to be $q_i, 1 \leq i \leq s - 1$, respectively. Therefore, $\Theta = \{\{l_i\}_{1 \leq i < s}, C, w_0, w_1, w_2, b_1, b_2\}$. The partial derivative of the quantizer $Q$ with respect to $l_i$
is computed by:
\[
\frac{\partial Q(x)}{\partial l_i} = \begin{cases} 
0, & \text{if } |x| < T_{i+1} \\
\text{sgn}(x) \prod_{j=1, j \neq i}^{k-1} l_j, & \text{if } T_k \leq |x| < T_{k+1}, k \geq i+1
\end{cases}
\]
(17)
where \( T_1 = \alpha_1, T_2 = \alpha_2 + (1 - \alpha_2)l_1, T_k = \alpha_k \prod_{j=1}^{k-2} l_j + (1 - \alpha_k) \prod_{j=1}^{k-1} l_j, \forall k \geq 2. \) Eq. (17) is used in Eq. (8) to calculate the sample proxy gradient with respect to \( l_i. \) Note that at the end of each epoch, each \( l_i \) is clipped by \([1 + \varepsilon, +\infty)\) with \( \varepsilon \) to be a small positive real number to satisfy the constraint \( l_i > 1, \) and the threshold set \( T \) is modified by the updated \( \{l_i\}_{i<s_c}. \) The channel value \( C \) is initialized to a positive number \( C_0 \) with \( C_0 \geq \alpha_1 \) to guarantee that the magnitude of each message in the first variable node update layer \( r^{(1)} \) is \( L_1. \) The sample proxy gradient with respect to \( C \) is obtained in the same manner as computing the sample proxy gradients with respect to weights and biases. The initial linear FAID defined by \( \{q_i\}_{i<s_c}, S, \) and \( C_0 \) needs to satisfy Property 1. Motivated by the fact that applying different learning rates to \( \Theta \) can help training convergence, we employ ADAM with mini-batches as optimizer. The ADAM takes the current and previous sample proxy gradients with respect to \( \Theta \) as inputs and adjusts the learning rates based on these sample proxy gradients. This process can be expressed as:
\[
\theta^{t+1} = \theta^t - \eta_{\text{ADAM}} \gamma(\theta^t, \gamma(\theta^{t-1}), ..., \gamma(\theta^0)), \forall \theta \in \Theta,
\]
where \( g_{\text{ADAM}} \) is the core function in ADAM that adjusts \( \eta \) for \( \theta \) based on \( \gamma(\theta^t), \gamma(\theta^{t-1}), ..., \gamma(\theta^0). \) To make the training reduce the expected loss, \( g_{\text{ADAM}}(\gamma(\theta^t), \gamma(\theta^{t-1}), ..., \gamma(\theta^0)) \) should satisfy Property 2, namely,
\[
\langle g_{\text{ADAM}}(\gamma(\theta^t), \gamma(\theta^{t-1}), ..., \gamma(\theta^0)) \rangle, \forall \theta \in \Theta
\]
Note that the training will not begin until the BER loss function is nonzero. Once the training starts, even if the current BER loss function is zero, the trainable parameters will still be updated.

To construct a training set, we randomly generate samples with a fixed \( p. \) The choice of \( p \) can neither be too large as the training will diverge if there is too much noise, nor too small as the training will wait for a long time to start due to the fact that most of the time the BER loss function is zero. A heuristic approach to search for a good \( p \) as described is follows. Consider the frame error rate (FER) decoding performance of the floating-point min-sum decoder with the same number of iterations as that of the RQNN, and denote the FER at \( p \) by \( P_{\text{FER}}(p). \) Let the size of mini-batches be \( S_m. \) We take the \( p \) that satisfies the following two conditions: 1) \( p \) is in the waterfall region; 2) \( P_{\text{FER}}(p) \cdot S_m \geq 1. \) The first condition indicates that when we use this \( p \) to construct a training set, some noisy configurations in the training set can be corrected by certain decoding rules, and the training is possible to converge. The second condition shows that if we use the floating-point min-sum decoder to decode \( S_m \) random samples from this training set, on average one sample may be decoded incorrectly. This implies that the BER loss function in the first epoch is likely to be nonzero, as the training begins with a quantized min-sum decoder. Therefore, the training can start soon. Generally speaking, we can construct a training set consisting of noisy realizations randomly collected with a set of different \( p, \) or we can train a linear FAID for every \( p \) in this set separately and choose one of these trained linear FAIDs for decoding under different channel conditions. Surprisingly, the numerical results show that if we conduct training on a proper \( p \) (for instance, use the above heuristic approach to select \( p)), \) the trained linear FAID can be generalized well to other unseen crossover probabilities.

The ADAM simultaneous training with STEs is summarized in Algorithm 1, where the \( t \)-th mini-batch is denoted as \( \{Y^{(t)}, X^{(t)}\}, \) and \( N_s \) is the number of mini-batches. \( N_s \) can be chosen to be the number of epochs when the training converges. Note that the gradients are accumulated in full precision, and the quantized activation values are used in both forward and backward propagations.

We remark that the order in which to train the levels in \( \mathcal{M} \) and the parameters in the RQNN (channel value \( C \), weights, and biases) affects the optimality of the local minimum. In this work we train all the parameters simultaneously. An alternative approach is to train the levels in \( \mathcal{M} \) and other parameters separately. In this approach, we first train the levels on a floating-point pre-trained model which is derived by training a floating-point recurrent NN, and then re-train the parameters in the pre-trained model to adjust to the levels. Comparing these two training strategies, it turns out that the simultaneous training has larger possibility to reach a good local minimum, since in the second training strategy, a floating-point model is first trained, which puts more constraints on the subsequent training and limits the capability to explore the searching space.

V. NUMERICAL RESULTS

In this section, we evaluate the BER/FER performance of the RQNN framework over the BSC. We build the RQNN framework in Python3.7 and use Pytorch library for training. We design linear FAIDs with 3-bit messages to show the design flexibility in terms of code length and column weight, one for a short code, Tanner code (155, 64) with column weight three, and the other for a medium length code, QC-LDPC code (1296, 972) with column weight four. These two codes are available in [23]. The simulation of \( D_{\text{FAID}} \) is carried out with the integer-valued LUTs. For comparisons with other conventional iterative decoders, we conduct simulations of MS decoding, Offset MS (OMS) decoding, and Sum Product Algorithm (SPA) decoding, which are implemented with floating-point precision. We provide numerical analysis in terms of the decoding performance with the same number of iterations, convergence, thresholds, and generalizations to longer code and more iterations.

A. Tanner Code with \( d_v = 3 \)

In the first example, we consider a column-weight 3 low rate Tanner code (155, 64) and design a 3-bit linear FAID for it via the joint training scheme.
Input: minibatches 
(\(Y^{(t)}, X^{(t)}\))\(_{1\leq i \leq N_s} \), \(S\), \(C_0\), \(\{q_i\}_{1\leq i \leq s}\), \(\varepsilon\), \(\eta\)

1. Initialization:

2. \(C \leftarrow C_0\), \(l_i \leftarrow q_i\), \(1 \leq i \leq s\),
3. set \(\mathcal{M}\) via Eq. (16) with \(\{l_i\}_{1\leq i \leq s}\),
4. set \(T\) via Eq. (4) with \(S\) and \(\mathcal{M}\), and
5. set \(Q\) via Eq. (1) with \(T\) and \(\mathcal{M}\);
6. for \(t = 1\) to \(N_s\) do /* t-th mini-batch \((Y^{(t)}, X^{(t)})\) */
7. Forward propagation:
8. Compute each layer via Eq. (2),
9. \(U \leftarrow \mathcal{D}_{RQNN}((-1)^{Y^{(t)}}T)\)
10. end
11. Backward propagation:
12. Set to zero all the gradients,
13. \(\Lambda \leftarrow \sum_{u \in U, x \in X^{(t)}} \Gamma(u, x);\)
14. /* compute the sample proxy gradients */
15. for \(\forall \theta \in \Theta\) do
16. /* \(\Theta = \{\{l_i\}_{1\leq i \leq s}, C, w_0, w_1, w_2, b_1, b_2\}\) */
17. Use Eq. (14) and Eq. (15) to compute \(\gamma(\theta^t)\) via Eq. (8)
18. end
19. Parameter Updates: /* ADAM optimizer */
20. for \(\forall \theta \in \Theta\) do
21. /* \(\Theta = \{\{l_i\}_{1\leq i \leq s}, C, w_0, w_1, w_2, b_1, b_2\}\) */
22. \(\theta^{t+1} \leftarrow \theta^t - \eta_{ADAM} \Gamma(\theta^t, \gamma(\theta^{t-1}), ..., \gamma(\theta^t))\)
23. end
24. for \(i = 1\) to \(s - 1\) do
25. \(l_i^{t+1} \leftarrow \text{clip}(l_i^{t+1}, [1 + \varepsilon, +\infty))\)
26. end
27. Use \(S\) and Eq. (4) to update \(T\)
28. end
29. Output: \(\mathcal{D}_{RQNN}\)

Algorithm 1 ADAM Simultaneous Training with STEs

We first construct a training set of 5000 random error patterns collected with \(p = 0.03\). (As shown in Fig. 3, for floating-point MS decoding, the waterfall begins around \(p = 0.04\), thus \(p = 0.03\) is in the waterfall region). The size of each mini-batch is 50, and the maximum number of iterations is set to be 5. Note that \(P_c, \text{FER}(p) \cdot S_m \approx 1.8 > 1\). In the initialization, \(C\) is set to be 2, and the scalar set \(S = \{1, 0, 0\}\), with \(\varepsilon = 0.1\).

We use a learning rate of 0.01 to train the RQNN, and it converges within 50 epochs. We remark that \(\varepsilon\) is not very sensitive to local minima, as the initial values of \(\{l_i\}\) are far from 1. The trained \(\mathcal{D}_{RQNN}\) has parameters listed in Table I. Based on these parameters, we convert the \(\mathcal{D}_{RQNN}\) into a 3-bit linear FAID \(\mathcal{D}_{FAID}\) consisting of one 2-dimensional array and one 3-dimensional array.

B. Medium-Length Code with \(d_v = 4\)

In the second example, we consider a column-weight 4 high rate QC LDPC code (1296, 972) and design a 3-bit linear FAID for it via the joint training scheme.

We first construct a training set of 5000 random error patterns collected with \(p = 0.008\). (As shown in Fig. 4, for floating-point MS decoding, the waterfall begins around \(p = 0.008\)). The size of each mini-batch is 10, and the maximum number of iterations is set to be 5. Note that \(P_c, \text{FER}(p) \cdot S_m \approx 4.3 > 1\). In the initialization, \(C\) is set to be 2, and the scalar set \(S = \{1, 0, 0\}\), with \(\varepsilon = 0.1\). By Eq. (4), \(T = \mathcal{M}\). The alphabet set \(\mathcal{M}\) has three levels, with \(\{l_1, l_2\}\) are initialized by \(\{2, 2\}\), respectively, i.e., \(M\) is initialized by \(\{1, 2, 4\}\). We use a learning rate of 0.005 to train the RQNN, and it converges within 50 epochs. We remark that since the initial values of \(\{l_i\}\) are far from 1, \(\varepsilon\) is not very sensitive to local minima. The trained \(\mathcal{D}_{RQNN}\) has parameters listed in Table I. Based on these parameters, we convert the \(\mathcal{D}_{RQNN}\) into a 3-bit linear FAID \(\mathcal{D}_{FAID}\) consisting of one 3-dimensional array and one 4-dimensional array.

C. Analysis and Discussion

1) Decoding Performance with the Same Number of Iterations: Fig. 3 and Fig. 4 show the FER decoding performance
within 5 iterations of various decoders for Tanner code (155, 64) and QC LDPC code (1296, 972), respectively. Fig. 3 compares the FER decoding performance of 3-bit $D_{\text{FAID}}^\text{RQNN}$ and floating-point BP algorithms including MS, OMS (offset $\lambda = 0.3$), and SPA for Tanner code (155, 64). We also include a 3-bit linear FAID (See [12], Example 2) for reference. Within 5 iterations, $D_{\text{FAID}}^\text{RQNN}$ performs as well as the floating-point OMS and SPA, and can achieve a FER one order of magnitude lower than that of MS. Fig. 4 demonstrates the FER performance of 3-bit $D_{\text{FAID}}^\text{RQNN}$, floating-point MS, OMS (offset $\lambda = 0.46$), and SPA for QC LDPC code (1296, 972). Within 5 iterations, $D_{\text{FAID}}^\text{RQNN}$ surpasses MS, OMS and SPA. At $p = 0.004$, it achieves a FER four orders of magnitude lower than that of MS, one order of magnitude lower than that of OMS, and less than one-third of SPA's FER.

2) Convergence: Fig. 5 shows the FER decoding performance of various decoders with different numbers of iterations for QC LDPC code (1296, 972). Compared to OMS (offset $\lambda = 0.46$), the 3-bit $D_{\text{FAID}}^\text{RQNN}$ with 7 iterations surpasses OMS with 10 iterations, and with 10 iterations it has similar decoding performance as OMS with 15 iterations. Compared to SPA, $D_{\text{FAID}}^\text{RQNN}$ with 15 iterations outperforms SPA with 30 iterations for FER less than $10^{-4}$ at much lower complexity. Fig. 6 demonstrates the FER performance versus the number of iterations at $p = 0.008$. To achieve FER less than $10^{-5}$, $D_{\text{FAID}}^\text{RQNN}$ requires 10 iterations, while OMS and SPA require 15 and 25 iterations, respectively. Among three decoders, $D_{\text{FAID}}^\text{RQNN}$ has the fastest convergence.

3) Decoding Thresholds: In Table II, we compare the decoding thresholds of various decoders for Tanner code (155, 64) and QC LDPC code (1296, 972), where “fp” indicates the floating-point precision. The decoding thresholds are computed via density evolution, which predict the ensembles’ behavior in waterfall region. For both codes, $D_{\text{FAID}}^\text{RQNN}$ has a decoding threshold comparable to that of OMS. For finite length LDPC codes, density evolution is asymptotic and does not take the code structure into account. On the contrary, the RQNN framework is built based on specific Tanner graph, thus although $D_{\text{FAID}}^\text{RQNN}$ has slightly worse decoding threshold than SPA, it still has good decoding performance in waterfall region. For example, for Tanner code (155, 64), the linear FAID in [12] has the largest decoding threshold in Table II, but it performs no better than $D_{\text{FAID}}^\text{RQNN}$.

4) Generalizations: The RQNN framework can be generalized to various settings, which have not been seen during training. The first generalization is for the crossover probability $p$. As mentioned in the experiments setup, we only collect error patterns at a specific $p$ for training. In the inference stage, we test $D_{\text{FAID}}^\text{RQNN}$ on a set of crossover probabilities, and the decoding performance is consistent over various $p$, as shown in Fig. 3 to Fig. 8. Since the RQNN framework has recurrent structure, we are able to evaluate the decoding performance of $D_{\text{FAID}}^\text{RQNN}$ with more iterations. Fig. 7 and Fig. 8 demonstrate the FER decoding performance of $D_{\text{FAID}}^\text{RQNN}$ and SPA with more iterations for Tanner code (155, 64) and QC LDPC code (1296, 972),...
TABLE II

Decoding Thresholds ($p^*$) of Various Decoders for Two Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>SPA (fp)</th>
<th>OMS (fp)</th>
<th>$\Phi_{\text{RQNN},\text{FAID}}$ (3-bit)</th>
<th>Linear FAID [12] (3-bit)</th>
<th>MS (fp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tanner code (155, 64)</td>
<td>0.114</td>
<td>0.109 (A = 0.3)</td>
<td>0.102</td>
<td>0.180</td>
<td>0.095</td>
</tr>
<tr>
<td>QC LDPC code (1296, 972)</td>
<td>0.03</td>
<td>0.026 (A = 0.46)</td>
<td>0.027</td>
<td>-</td>
<td>0.023</td>
</tr>
</tbody>
</table>

Since the RQNN framework takes the specific code structure into account, we investigate the generalization to longer code with similar code structure. Consider the Tanner code (310, 128), which has the same dominant trapping set as Tanner code (155, 64). We evaluate the FER performance within 100 iterations of $\Phi_{\text{RQNN},\text{FAID}}$ designed for Tanner code (155, 64), linear FAID [12], and SPA for Tanner code (310, 128) in Fig. 7. Both 3-bit decoders approach SPA and surpass SPA for FER less than $10^{-4}$, and $\Phi_{\text{RQNN},\text{FAID}}$ is better than the linear FAID [12].

It is easy to see that each LUT is symmetric as it is invariant to the ordering of incoming messages, thus we only need to save part of its entries. We can even replace each level $L_i$ by integer $i$ to simplify the implementation. Table III shows the 3-bit $\Phi_{\text{RQNN}}$ of $\Phi_{\text{RQNN},\text{FAID}}$ of QC LDPC code (1296, 972), where $m_{i,k}$ indicates that $i_2 = M_j$, $i_3 = M_k$. As it is shown in [24], [25], a decoder supporting multiple decoding rules can be efficiently implemented in hardware, thus $\Phi_{\text{RQNN},\text{FAID}}$ requires only a small hardware overhead.

D. Relation with Previous Related Work

There have been significant efforts on designing FAIDs of LDPC codes for different channel models. Most of them have performance approaching BP decoding. For example, for AWGNC, Nguyen-Ly et al. [26] used density evolution to optimize the FAIDs, and showed that 4-bit precision FAIDs can approach the BP decoding performance. Lewandowsky et al. [27] obtained similar results via Information Bottleneck method. Romero et al. [28] got similar results for both AWGNC and BSC by utilizing an optimal quantizer at each step of density evolution to maximize mutual information. The only work capable of outperforming BP decoding is in [12], where Planjery et al. constructed FAIDs for column-weight-three regular LDPC codes to correct trapping sets. They showed that the 3-bit precision FAIDs surpassed BP and all other message passing decoders in error floor region.

In this work, we show the capability of neural networks on designing FAIDs with low precision over BSC with competitive decoding performance. Compared with previous works, the RQNN framework has several advantages. As opposed to the prevalent approach of designing the message update rules by trial and error, where for each candidate update rule, its threshold is found by the density evolution and then optimized by varying the update rule, the NN method designs the update rules automatically. Our framework results in FAID optimized for both superior error correction performance as well as fast convergence in the waterfall region.
VI. CONCLUSION

In this paper, we proposed a new design of linear finite alphabet iterative decoding rules of LDPC codes over BSC via recurrent quantized neural networks. The RQNN framework can generate linear FAIDs with only 3-bit precision that perform as well as, and are capable of surpassing, floating-point OMS and SPA decoding. We provided a BER loss function for BSC, and introduced how to address zero gradient issue in backward propagation via STEs. We suggested a joint training scheme to simultaneously train the levels in alphabet set and the RQNN parameters. The numerical results show that the RQNN-aided linear FAIDs are capable of surpassing floating-point BP in waterfall region at much lower complexity and memory usage, faster convergence, and can be generalized well to more iterations and longer codes with similar code structure. Future research includes how to use neural networks to solve trapping set problems to design decoders with low error floor.

REFERENCES


TABLE III

| 3-BIT $\Phi_{RQNN}$ IN $D_{RQNN}$ FOR QC LDPC CODE (1296, 972) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\Phi_{RQNN}$ | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\phi_{11}$ | $\phi_{12}$ | $\phi_{13}$ |
| $\Phi_{FAID}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| $\Phi_{RQNN}$ | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $\phi_6$ | $\phi_7$ | $\phi_8$ | $\phi_9$ | $\phi_{10}$ | $\phi_{11}$ | $\phi_{12}$ | $\phi_{13}$ |
| $\Phi_{FAID}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

Authorized licensed use limited to: The University of Arizona. Downloaded on August 29,2020 at 20:33:33 UTC from IEEE Xplore. Restrictions apply.
Xin Xiao received the B.S. degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012. Under a dual degree program, she received the M.E. degree in system LSI from the Graduate School of Information, Production and System, Waseda University, Kitakyushu, Fukuoka, Japan, in 2013, and the M.S. degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2015. She is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Arizona, Tucson, AZ, USA.

Since 2016, she has been a Research Assistant with the Electrical and Computer Engineering Department, University of Arizona. Her current research interests are in the general area of error correction coding for communication and storage systems, and deep learning and optimization in channel coding.

Bane Vasić (Fellow, IEEE) is currently a Professor of electrical and computer engineering and mathematics with the University of Arizona and the Director of the Error Correction Laboratory. He is an inventor of the soft error-event decoding algorithm, and the key architect of a detector/decoder for Bell Labs data storage read channel chips which were regarded as the best in industry. His pioneering work on structured low-density parity-check (LDPC) error-correcting codes and the invention of codes has enabled low-complexity iterative decoder implementations. Structured LDPC codes are today adopted in a number of communications standards and data storage systems. He is known for his theoretical work in error correction coding theory and codes on graphs which has led to the characterization of the hard decision iterative decoders of LDPC codes, and design of decoders with best error-floor performance known today. He is a co-founder of Codelucida, a startup company developing advanced error correction solutions for communications and data storage. He is a Fulbright Scholar, da Vinci Fellow, and the past Chair of IEEE Data Storage Technical Committee.

Ravi Tandon (Senior Member, IEEE) received the B.Tech. degree in electrical engineering from IIT Kanpur in 2004, and the Ph.D. degree in electrical and computer engineering from the University of Maryland, College Park (UMCP), in 2010. From 2010 to 2012, he was a Post-Doctoral Research Associate at Princeton University. He is currently an Assistant Professor with the Department of ECE, University of Arizona. Prior to joining the University of Arizona in Fall 2015, he was a Research Assistant Professor at Virginia Tech with positions in the Bradley Department of ECE, Hume Center for National Security and Technology, and at the Discovery Analytics Center in the Department of Computer Science. His current research interests include information theory and its applications to wireless networks, communications, security and privacy, machine learning, and data mining. He is a recipient of the 2018 Keysight Early Career Professor Award, the NSF CAREER Award in 2017, and a Best Paper Award at IEEE GLOBECOM 2011. He currently serves as an Editor for IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS.

Shu Lin received the B.S.E.E. degree from the National Taiwan University, Taipei, Taiwan, in 1959, and the M.S. and Ph.D. degrees in electrical engineering from Rice University, Houston, TX, USA, in 1964 and 1965, respectively.

In 1965, he joined the Faculty of the University of Hawaii, Honolulu, as an Assistant Professor of electrical engineering. He became an Associate Professor in 1969 and a Professor in 1973. In 1986, he joined Texas A&M University, College Station, TX, as the Irma Runyon Chair Professor of electrical engineering. In 1987, he returned to the University of Hawaii. From 1978 to 1979, he was a Visiting Scientist at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, where he worked on error control protocols for data communication systems. He spent the academic year of 1996 to 1997 as a Visiting Professor at the Technical University of Munich, Munich, Germany. He retired from the University of Hawaii in 1999. He is currently an Adjunct Professor at the University of California, Davis, CA, USA. He has published numerous technical papers in IEEE Transactions and other refereed journals. He is the author of the book An Introduction to Error-Correcting Codes (Englewood Cliffs, NJ: Prentice-Hall, 1970). He has also co-authored (with D. J. Costello) the book Error Control Coding: Fundamentals and Applications (Upper Saddle River, NJ: Prentice-Hall, 1st edition, 1982, 2nd edition, 2004), (with T. Kasami, T. Fujiwara, and M. Fossorier) the book Trellises and Trellis-Based Decoding Algorithms (Boston, MA: Kluwer Academic, 1998), (with W. E. Ryan) the book Channel Codes: Classical and Modern (Cambridge University Press, 2009), and (with J. Li, K. Abdel-Ghaffar, W. E. Ryan, and D. J. Costello, Jr.) the book LDPC Code Designs, Constructions, and Unification (Cambridge University Press, 2017). He has served as the Principle Investigator on 32 research grants. His current research areas include algebraic coding theory, coded modulation, error control systems, and satellite communications.

Dr. Lin is a member of the IEEE Information Theory Society and the Communication Society. He has served as the Program Co-Chairman for the IEEE International Symposium of Information Theory held in Kobe, Japan, in June 1988. He was the President of the IEEE Information Theory Society in 1991. In 1996, he was a recipient of the Alexander von Humboldt Research Prize for U.S. Senior Scientists and a recipient of the IEEE Third-Millennium Medal, 2000. In 2007, he was a recipient of The Communications Society Stephen O. Rice Prize in the Field of Communications Theory. In 2014, he was awarded the NASA Exceptional Public Achievement Medal. He was awarded the 2019 Albert Nelson Marquis Lifetime Achievement Award. He is a recipient of the 2020 IEEE Leon K. Kirchmayer Graduate Teaching Award. He has served as the Associate Editor for Algebraic Coding Theory for IEEE TRANSACTIONS ON INFORMATION THEORY from 1976 to 1978.