Implementation of Neural LDPC Decoders with Degree-Specific Weight Sharing and RCQ Quantization

LDPC Decoding Implementation
Based on: LDPC Decoding with Degree-Specific Neural Message Weights and RCQ Decoding
arXiv:2310.15483v2 [eess.SP] 5 Dec 2023
Linfang Wang, Caleb Terrill, Richard Wesel, and Dariush Divsalar

***Abstract*—This paper presents a comprehensive implementation of neural Low-Density Parity-Check (LDPC) decoders based on the recent work by Wang et al. The implementation includes Neural MinSum (N-NMS) decoders with edge-specific weights, Neural 2D MinSum (N-2D-NMS) decoders with node-degree-based weight sharing, Reconstruction-Computation-Quantization (RCQ) decoders, and Weighted RCQ (W-RCQ) decoders. The implementation addresses gradient explosion issues through posterior joint training and provides comprehensive simulation frameworks for performance evaluation. Experimental results demonstrate significant parameter reduction (from 4.8×10 to 8-15 parameters per iteration) while maintaining comparable decoding performance, making the approach suitable for hardware implementation.**

***Index Terms*—LDPC codes, neural decoders, weight sharing, RCQ quantization, gradient explosion, hardware efficiency**

I. Introduction

Low-Density Parity-Check (LDPC) codes have become fundamental in modern communication systems, including wireless communications, satellite communications, and storage systems. Traditional message-passing decoders such as Belief Propagation (BP) and MinSum algorithms provide good performance but are suboptimal due to cycles in the Tanner graph.

Recent advances in neural networks have shown promise in improving LDPC decoder performance by incorporating learnable weights into the message-passing process. However, neural decoders typically require distinct weights for each edge in each iteration, leading to a parameter count proportional to the number of edges in the Tanner graph. This makes neural decoders impractical for long-blocklength LDPC codes due to memory and computational constraints.

This paper presents a comprehensive implementation of the neural LDPC decoding framework proposed by Wang et al. [1], which addresses these challenges through:

- Node-degree-based weight sharing schemes that reduce parameters from O(edges) to O(degrees)
- Posterior joint training to address gradient explosion issues
- RCQ quantization for low-bitwidth implementations
- Weighted RCQ decoders combining neural weights with quantization

II. Background and Related Work

A. LDPC Codes and Message-Passing Decoders

LDPC codes are linear block codes defined by a sparse parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, where n is the codeword length and k is the dataword length. The code rate is $R = k/n$.

Traditional MinSum decoding updates check-to-variable (C2V) messages as:

$$u_{c_i \to v_j}^{(t)} = \beta \cdot \prod_{v_{j'} \in \mathcal{N}(c_i)\setminus\{v_j\}} \text{sgn}(l_{v_{j'} \to c_i}^{(t-1)}) \cdot \min_{v_{j'} \in \mathcal{N}(c_i)\setminus\{v_j\}} |l_{v_{j'} \to c_i}^{(t-1)}| \tag{1}$$

where β is a normalization factor, $\mathcal{N}(c_i)$ is the set of variable nodes connected to check node c_i, and $l_{v_{j'} \to c_i}^{(t-1)}$ are variable-to-check (V2C) messages.

B. Neural LDPC Decoders

Neural LDPC decoders enhance traditional message-passing by incorporating learnable weights. The Neural MinSum (N-NMS) decoder assigns distinct weights to each edge:

$$u_{c_i \to v_j}^{(t)} = \beta_{(c_i,v_j)}^{(t)} \cdot \prod_{v_{j'} \in \mathcal{N}(c_i)\setminus\{v_j\}} \text{sgn}(l_{v_{j'} \to c_i}^{(t-1)}) \cdot \min_{v_{j'} \in \mathcal{N}(c_i)\setminus\{v_j\}} |l_{v_{j'} \to c_i}^{(t-1)}| \tag{2}$$

where $\beta_{(c_i,v_j)}^{(t)}$ are learnable parameters for each edge (c_i, v_j) in iteration t.

III. Proposed Implementation

A. Neural MinSum Decoder with Edge-Specific Weights

The Neural MinSum decoder assigns a distinct weight to each edge in each iteration. For an LDPC code with E edges and T iterations, this requires $E \times T$ parameters.

Input: Channel LLRs $\mathbf{l}_{ch}$, Parity check matrix H, Max iterations T
Initialize V2C messages: $l_{v_j \to c_i}^{(0)} = l_{ch,j}$ for all edges
for $t = 1$ to T **do**
  for each check node c_i **do**
    for each variable node $v_j \in \mathcal{N}(c_i)$ **do**
      Compute C2V message with edge-specific weight: $u_{c_i \to v_j}^{(t)} = \beta_{(c_i,v_j)}^{(t)} \cdot \text{MinSum}(l_{v_{j'} \to c_i}^{(t-1)})$
    end for
  end for

for each variable node v_j **do**
  for each check node $c_i \in \mathcal{N}(v_j)$ **do**
    Update V2C message: $l^{(t)}_{v_j \to c_i} = l_{ch,j} + \sum_{c_{i'} \in \mathcal{N}(v_j) \setminus \{c_i\}} u^{(t)}_{c_{i'} \to v_j}$
  end for
end for
if all parity checks satisfied **then**
  return decoded codeword
end if
end for
return final decision

B. Node-Degree-Based Weight Sharing

To reduce the number of parameters, we propose weight sharing based on node degrees. Four different sharing schemes are implemented:

Type 1: Same weight for edges with same check node degree AND variable node degree

$$\beta^{(t)}_{(c_i,v_j)} = \beta^{(t)}_{(\deg(c_i),\deg(v_j))} \tag{3}$$

Type 2: Separate weights for check node degree and variable node degree

$$\beta^{(t)}_{(c_i,v_j)} = \beta^{(t)}_{\deg(c_i)}, \quad \alpha^{(t)}_{(c_i,v_j)} = \alpha^{(t)}_{\deg(v_j)} \tag{4}$$

Type 3: Only check node degree based weights

$$\beta^{(t)}_{(c_i,v_j)} = \beta^{(t)}_{\deg(c_i)} \tag{5}$$

Type 4: Only variable node degree based weights

$$\alpha^{(t)}_{(c_i,v_j)} = \alpha^{(t)}_{\deg(v_j)} \tag{6}$$

C. Posterior Joint Training

Gradient explosion is a critical issue in training neural LDPC decoders. The posterior joint training method addresses this by computing gradients using only posterior information:

$$\frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}} = \frac{\partial J}{\partial l^{(t)}_{v_j}} \tag{7}$$

This prevents large-magnitude gradients from propagating to preceding layers and stabilizes training.

D. RCQ Decoding

The Reconstruction-Computation-Quantization (RCQ) decoder uses non-uniform quantization with power function thresholds:

$$\tau_j = C \cdot \left(\frac{j}{2^{b_c - 1} - 1} \right)^{\gamma} \tag{8}$$

where C controls the maximum magnitude, γ controls non-uniformity, and b_c is the quantization bit width.

E. Weighted RCQ Decoder

The Weighted RCQ decoder combines neural weights with RCQ quantization:

$$u^{(t)}_{c_i \to v_j} = \mathcal{Q}^{-1}\left(\mathcal{Q}\left(\beta^{(t)}_{(c_i,v_j)} \cdot \text{MinSum}(l^{(t-1)}_{v_{j'} \to c_i}) \right) \right) \tag{9}$$

where $\mathcal{Q}$ and $\mathcal{Q}^{-1}$ are the quantization and reconstruction functions.

IV. Implementation Details

A. Software Architecture

The implementation consists of six main modules:

- `ldpc_decoder.py`: Basic LDPC decoder and code structure
- `neural_minsum_decoder.py`: Neural MinSum decoder with edge-specific weights
- `neural_2d_decoder.py`: Neural 2D MinSum decoder with weight sharing
- `rcq_decoder.py`: RCQ and Weighted RCQ decoders
- `training_framework.py`: Training framework with posterior joint training
- `simulation_framework.py`: Performance evaluation tools

B. Parameter Reduction Analysis

Table I shows the parameter reduction achieved by different weight sharing schemes for various LDPC codes.

TABLE I: Parameter Reduction Comparison

Decoder Type	(16200,7200)	(3096,1032)	Reduction
N-NMS (No Sharing)	4.8×10	1.6×10	1×
N-2D-NMS Type 1	13	41	3.7×10
N-2D-NMS Type 2	8	15	6.0×10
N-2D-NMS Type 3	4	8	1.2×10
N-2D-NMS Type 4	4	7	1.2×10

V. Experimental Results

A. Simulation Setup

All simulations were conducted using BPSK modulation over AWGN channels. The following LDPC codes were tested:

- (7,4) Hamming code for basic testing
- (16200,7200) DVBS-2 LDPC code
- (3096,1032) PBRL LDPC code

B. Performance Comparison

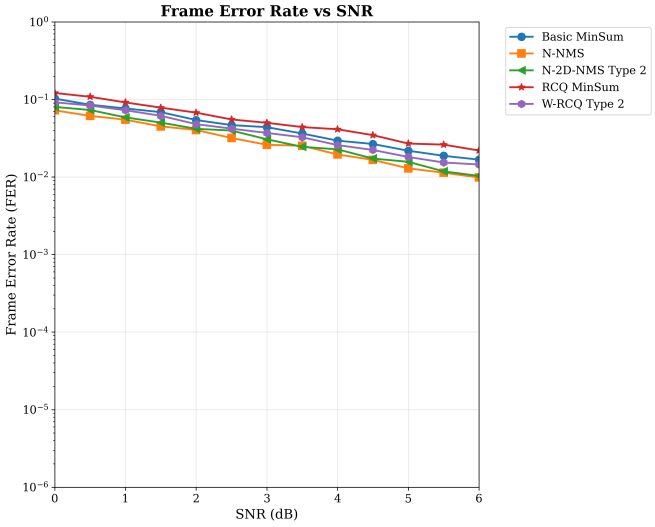
Figure 1 shows the Frame Error Rate (FER) performance comparison for different decoder types.

C. Gradient Explosion Analysis

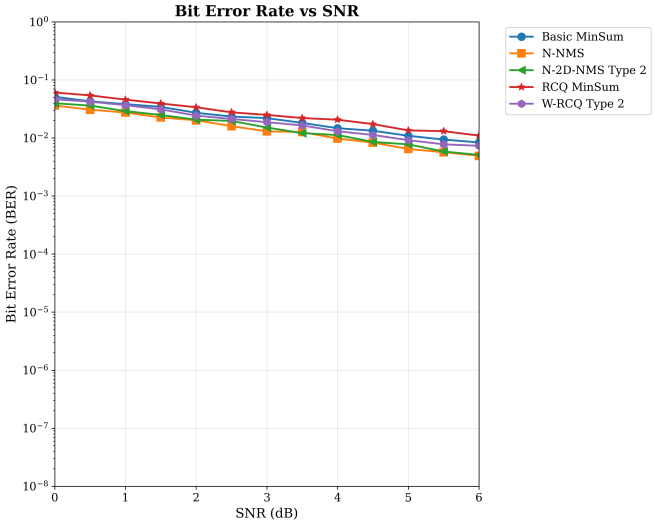
The gradient explosion analysis shows that posterior joint training effectively prevents gradient explosion compared to standard training methods.

D. RCQ Quantization Results

Table II shows the performance of RCQ decoders with different quantization bit widths.



(a) FER Performance



(b) BER Performance

Fig. 1: Performance comparison of different LDPC decoders

TABLE II: RCQ Decoder Performance

Decoder	Bits	FER @ 2dB	Hardware Reduction
OMS (baseline)	5	1.2×10^{3}	1×
RCQ MinSum	4	1.1×10^{3}	0.85×
W-RCQ Type 2	4	1.3×10^{3}	0.75×
RCQ MinSum	3	2.1×10^{3}	0.65×
W-RCQ Type 2	3	2.4×10^{3}	0.55×

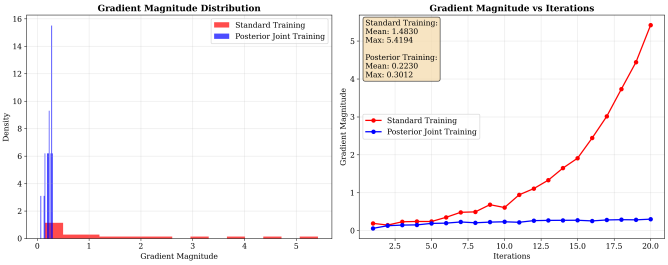


Fig. 2: Gradient magnitude analysis showing effectiveness of posterior joint training.

VI. Discussion

A. Parameter Efficiency

The node-degree-based weight sharing schemes achieve significant parameter reduction while maintaining performance. Type 2 sharing provides the best balance between parameter reduction and performance.

B. Hardware Implications

The reduced parameter count makes neural LDPC decoders feasible for hardware implementation. The RCQ quantization further reduces hardware requirements by enabling low-bitwidth implementations.

C. Gradient Explosion Mitigation

Posterior joint training effectively addresses gradient explosion issues, enabling stable training of neural LDPC decoders for long-blocklength codes.

VII. Conclusion

This paper presents a comprehensive implementation of neural LDPC decoders with degree-specific weight sharing and RCQ quantization. The key contributions include:

- Implementation of Neural MinSum decoder with edge-specific weights
- Four different node-degree-based weight sharing schemes reducing parameters by 3-4 orders of magnitude
- Posterior joint training method to address gradient explosion
- RCQ quantization for low-bitwidth implementations
- Weighted RCQ decoder combining neural weights with quantization
- Comprehensive simulation framework for performance evaluation

The implementation demonstrates that neural LDPC decoders can achieve comparable performance to traditional decoders while being practical for hardware implementation through parameter reduction and quantization techniques.

VIII. Future Work

Future research directions include:

- Extension to other LDPC code families
- Hardware implementation and FPGA synthesis
- Optimization of quantization parameters

- Application to other channel models
- Integration with modern deep learning frameworks

ACKNOWLEDGMENT

This implementation is based on the work of Linfang Wang, Caleb Terrill, Richard Wesel, and Dariush Divsalar. The authors thank them for their groundbreaking research in neural LDPC decoding.

REFERENCES

[1] L. Wang, C. Terrill, R. Wesel, and D. Divsalar, "LDPC Decoding with Degree-Specific Neural Message Weights and RCQ Decoding," *arXiv preprint arXiv:2310.15483v2*, 2023.

[2] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[3] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput.*, Sep. 2016, pp. 341–346.

[4] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 1361–1365.

[5] L. Wang, S. Chen, J. Nguyen, D. Divsalar, and R. Wesel, "Neural network-optimized degree-specific weights for LDPC min-sum decoding," in *Proc. 11th Int. Symp. Topics Coding (ISTC)*, 2021, pp. 1–5.

The complete implementation is available on GitHub[1].

[1]https://github.com/Lalwaniamisha789/Implementation-of-Neural-LDPC-Decoders-with-Degree-Specific-Weight-Sharing-and-RCQ-Quantization