Graph Neural Networks for Channel Decoding

Introduction

- In communication systems, reliable data transmission is essential
- Noise, interference, and signal degradation can lead to errors and data loss
- Channel coding, or error control coding, allows the receiver to detect and correct errors
- Among error correction codes LDPC and BCH are very popular
- LDPC codes use a sparse parity-check matrix and decodes close to the Shannon limit
- BCH codes can detect and correct multiple random errors
- However traditional decoding methods have limitations, especially for short codewords
- Belief Propagation (BP) decoding works well for long codewords
- Maximum Likelihood (ML) decoding has exponential complexity
- Graph Neural Networks (GNNs) are a promising solution to these challenges

Introduction

- GNNs are specialized neural networks that process data represented as graphs
- In the context of channel coding, the parity-check matrix is represented as a graph
- The nodes represent variables (bits) and constraints represent the edges
- This representation allows GNNs to perform message passing over the graph, similar to BP
- GNN is trained to learn a generalized message passing algorithm
- Instead of using fixed functions to update messages, the GNN learns the functions from data
- Learning enables the GNN adapt to complex structures and outperform traditional methods
- Using the Cora dataset, a node classification task I illustrate GNN functionality
- A simple GNN model for LDPC decoding is developed using PyTorch
- Key findings from NVIDIA's paper (Cammerer et al. 2022) are reimplemented

Related Work

- LDPC codes introduced by (Gallager 1962) were among the first codes to allow efficient error correction
- GD Forney introduced the concept of representing error-correcting codes using graphs
- (Richardson and Urbanke 2008) presented LDPC and other modern coding theories
- (Chung et al. 2001) illustrated the potential of LDPC codes designed close to Shannon limit
- (Varadarajulu et al. 2024) investigated GNN for BCH codes
- (Honkala, Korpi, and Huttunen 2021) used convolutional learning receiver for end-to-end decoding
- (Clausius et al. 2024) proposed joint equalization and decoding using GNNs in correlated wireless channels
- (Liao et al. 2023) applied GNNs to the construction of polar codes using its graph structure
- (Wu et al. 2023), proposes a method that uses shared parameters in the GNN decoding
- The present work builds on these advancements, integrating GNNs within LDPC decoding

Problem Definition

- Parity-Check Matrix (H)
  - A fundamental matrix defines the structure of LDPC and BCH code
  - Each row represents a parity-check constraint and each column corresponds to a bit in the codeword
  - Mathematically, a binary vector c represents a valid codeword if it satisfies $\mathbf{Hc^T = 0}$
  - Here H is the parity-check matrix, c is the codeword vector and 0 is the zero vector
  - This equation ensures that the codeword satisfies all parity-check constraint in a Galois Field (GF)

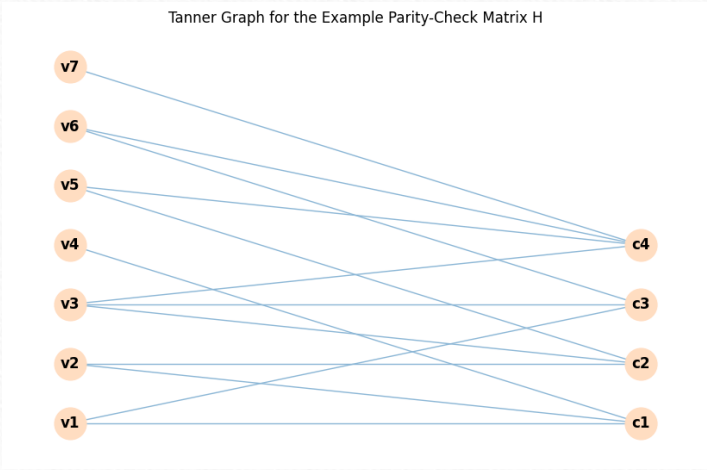
$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

- **First row**: $v_1 + v_2 + v_4 = 0 \pmod 2$
- **Second row**: $v_2 + v_3 + v_5 = 0 \pmod 2$
- **Third row**: $v_1 + v_3 + v_6 = 0 \pmod 2$
- **Fourth row**: $v_3 + v_5 + v_6 + v_7 = 0 \pmod 2$

Problem Definition

- Tanner Graph
  - Bipartite graph represents the code structure
  - Variable nodes correspond to transmitted bits and check nodes represent parity-check constraints
  - Edges in the graph indicate connections dictated by non-zero elements in the parity-check matrix
  - The Tanner graph is formally defined as G = (V,C,E)
  - Here V is the set of variable nodes (bits of the codeword)
  - C is the set of check nodes (parity-check constraints)
  - E is the set of edges connecting variable nodes to check nodes

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$



Problem Definition

- Log-Likelihood Ratio (LLR):
  - A measure used to quantify the probability of each bit being a 0 or 1 based on received noisy data
  - The LLR is used to estimate the most likely transmitted bit
  - A positive LLR suggests x = 0, and a negative LLR suggests x = 1
  - The LLR is derived from the observed noisy signal y and is given by:

$$\mathrm{LLR}(y) = \log\left(\frac{P(y|x=0)}{P(y|x=1)}\right)$$

  - Here P(y|x = 0) is the likelihood of receiving y given the transmitted bit x = 0
  - P(y|x = 1) is the likelihood of receiving y given the transmitted bit x = 1

Problem Definition - Decoding Process

- The GNN is trained on examples of transmitted and received data
- This allows it to learn message-passing rules for error correction
- Once trained, the GNN takes in noisy received data and applies its learned rules
- It estimates the most probable original message by iteratively updating estimates
- The decoding process specifically follows
    1. **Initialization with LLR :** LLR values provide initial estimate for each VN in the Tanner graph
    2. **Message Passing in the Tanner Graph :**
        i. Variable nodes send updated estimates to connected check nodes based on their LLR and incoming messages.
        ii. Check nodes send messages back to variable nodes, representing the probability that each bit satisfies the parity-check
    3. **Parity-Check Constraints (Matrix H):** Throughout iterations, variable nodes adjust estimates to satisfy the parity-check constraints
    4. **Final Decision :** After iterations, a hard decision is made for each bit based on the final LLR values

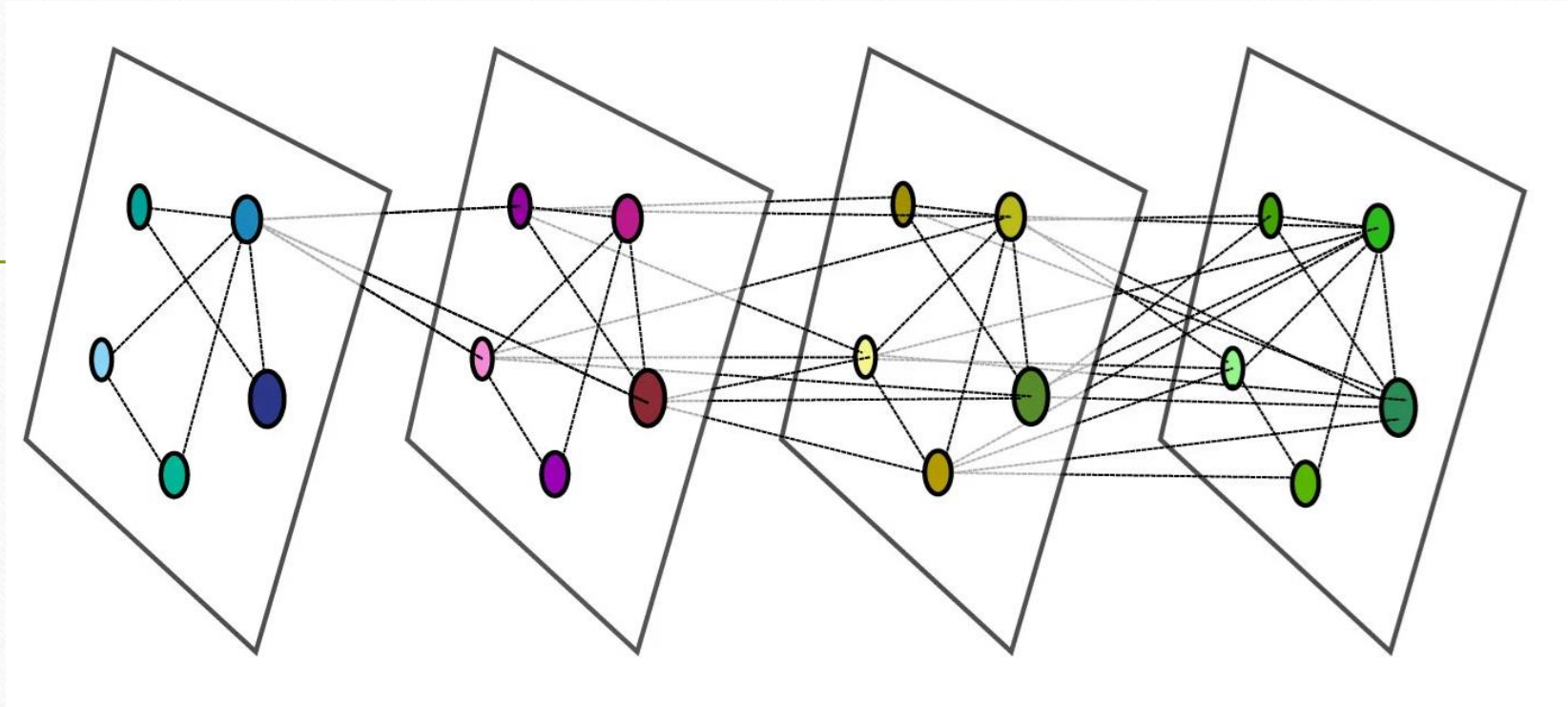
Methodology

- Basic Concepts of GNNs
  - Initial efforts focused on understanding the foundational principles of GNNs
  - GNNs are neural networks designed for graph-structured data
  - Each graph G consist of nodes V and edges E
  - Nodes in a GNN have feature vectors that store their information
  - Nodes communicate with neighbors to update their feature representations
  - A node gathers and aggregates information from its neighbors and updates its own feature vector

$$h_i^{(t+1)} = Update\Big(h_i^{(t)},\ Aggregate\Big(\{Message(h_i^{(t)}, h_j^{(t)}) \mid j \in \mathcal{N}(i)\}\Big)\Big)$$

where:

- $h_i^{(t)}$: the state (or feature) of node v_i at iteration t.
- $\mathcal{N}(i)$: the neighbors of v_i.
- Message, Aggregate, and Update are functions defined by the GNN architecture.



Message Passing Visualization in a 5-Node GNN. Each node aggregates information from its neighbors over multiple iterations

Experiments

- GNN operation was examined using both PyTorch Geometric's built-in MessagePassing class and a manual PyTorch approach.
- This laid a solid foundation for more complex Cora classification and LDPC decoding tasks
- A simple classification task was realized using the Cora dataset (McCallum et al. 2000),
- Cora dataset is a widely used benchmark for graph learning
- Each node represents a scientific paper with features derived from word occurrences
- Edges in the Cora dataset represent citation relationships
- The goal was to classify nodes by category based on their features and neighboring nodes
- This was realized using a simple Graph Convolutional Network model
- The GCN model includes two layers :
  - the first layer applies ReLU activation after aggregating neighbor features
  - the second layer uses log softmax for node classification

Basic LDPC code decoding using GNN

- A preliminary experiment explored GNN application to LDPC codes
- A Tanner graph was constructed from an LDPC parity-check matrix H
- The GNN iteratively refines predictions of codeword bits by message passing between connected nodes
- This simulated the Belief Propagation in GNN
- Raw values of the received noisy bits served as input features, without using LLRs
- During training, noise is added to codewords to simulate errors

Table 1: Decoded Codewords with True and Predicted Values, Iterations, MSE, and Bit Mismatches

True Codeword	Predicted Codeword	Iteration	MSE	Bit Mismatches
[1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1]	[1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1]	1	0.0304	0
[0 1 0 1 0 1 0 0 1 0 1 1 0 0 1 1]	[0 1 0 1 0 1 0 0 1 0 1 1 1 0 1 1]	2	0.0634	1
[0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 1]	[0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 1]	1	0.0683	0
[0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1]	[0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 1]	1	0.1259	1
[0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0]	[0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0]	1	0.0357	0
[0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0]	[0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0]	1	0.0377	0
[1 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1]	[0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1]	1	0.0774	1
[0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0]	[0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0]	1	0.1493	1
[1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1]	[1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0]	2	0.1109	1
[1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0]	[1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0]	1	0.1065	1

Reimplementation of Paper results

- Key packages like TensorFlow, PyTorch, and Sionna were utilized
- Sionna's library from NVIDIA enables efficient channel simulation and LDPC decoding
- An end-to-end model was created to simulate the communication pipeline
- The model integrates components like a Gaussian Prior Source and QAM Mapper/Demapper for symbol-based decoding
- Baseline decoding was implemented, starting with an uncoded scheme for initial BER/BLER reference
- A weighted BP model was developed to optimize BP decoding by introducing trainable weights
- Following this The GNN based BP decoder, the core of this project, was realized
- The procedure replicated the training structure from the original paper but scaled for reduced computational needs

Experimental Results and Result Analysis

- GNN-based LDPC decoder model is trained using a staged approach across three training phases
- Each training phase uses decreasing learning rates for gradual optimization
- First phase operates at 5.0 dB SNR and increases by 0.5 dB for each subsequent stage
- Authors' model underwent 635,000 training iterations across multiple phases, the reduced setup employed only 20,000 iterations

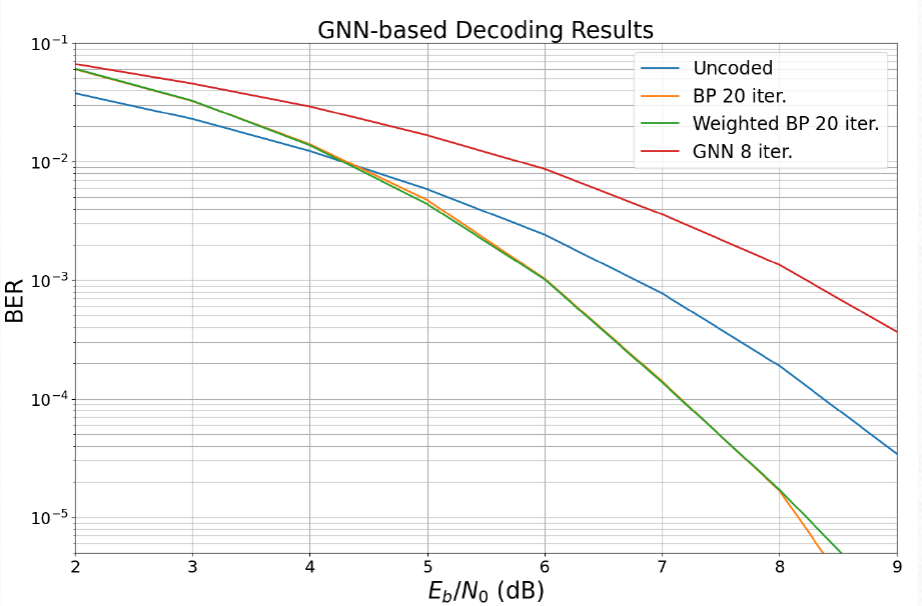


Table 2: Comparison of BER and BLER between Reduced Training and Original Training Setups

EbNo [dB]	BER (Mine)	BER (Authors)	Diff. BER	BLER (Mine)	BLER (Authors)	Diff. BLER	Comments
2.0	6.66e-02	5.78e-02	+1.88e-02	9.89e-01	7.55e-01	+2.33e-01	Slightly higher BER/BLER in the reduced training setup, but results are still quite close to the authors' model. This indicates the GNN's ability to retain decoding effectiveness even with fewer training iterations at low SNRs.
3.0	4.55e-02	2.87e-02	+1.68e-02	9.45e-01	4.58e-01	+4.87e-01	A more noticeable increase in BER/BLER as SNR improves. The results suggest that while reduced training provides reasonable decoding, the model could benefit from additional training for improved accuracy at this SNR level.
4.0	2.92e-02	1.01e-02	+1.91e-02	8.47e-01	1.90e-01	+6.56e-01	At 4 dB, BER and BLER show increased differences, highlighting the impact of limited training. However, the model maintains robustness in decoding, indicating practical viability despite higher error rates in the mid-SNR range.
5.0	1.67e-02	2.01e-03	+1.47e-02	6.57e-01	4.52e-02	+6.12e-01	The reduced training model shows significantly higher BER/BLER compared to the authors' model. This suggests a trade-off, where reduced training saves resources but impacts performance. Still, it remains viable for moderate SNR applications.
6.0	8.67e-03	2.28e-04	+8.44e-03	4.21e-01	6.12e-03	+4.15e-01	As SNR increases, the gap between reduced and original training becomes more pronounced. Additional training iterations could potentially narrow this gap, especially for applications requiring high accuracy at elevated SNR levels.
7.0	3.60e-03	1.69e-05	+3.58e-03	2.03e-01	5.75e-04	+2.03e-01	Higher error rates with reduced training indicate potential room for improvement, especially in high-SNR ranges. Further fine-tuning or adaptive learning could enhance decoding reliability under these conditions.
8.0	1.34e-03	1.87e-06	+1.34e-03	8.11e-02	8.60e-05	+8.10e-02	At this SNR, the performance gap continues to be significant. The reduced training setup achieves moderate success, though further optimization is needed for high-precision applications in this range.
9.0	3.67e-04	2.38e-07	+3.67e-04	2.29e-02	1.40e-05	+2.28e-02	A large performance gap exists at 9 dB, suggesting that reduced training may not fully capture the required complexity at high SNR levels. Despite this, the model demonstrates proof-of-concept efficiency for simpler decoding tasks with lower computational demand.

Conclusion and Future Directions

- The project explores the application of GNNs to the decoding of error correcting codes
- GNN message-passing mechanisms and their adaptability to channel coding challenges were explored
- The project shows that GNNs can leverage the structural properties of LDPC codes as seen in the Tanner graph representation to enhance decoding accuracy and speed
- The reimplementation demonstrated that even with reduced training iterations, the GNN-based decoder maintained competitive performance
- Integrating additional GNN architectures, such as Graph transformers or attention-based models, could enhance decoding accuracy
- The GNN framework can be extended to other structured codes, such as BCH and polar codes
- Further investigating unsupervised or self-supervised training strategies can further reduce the need for extensive labeled data