LDPC Code implementation through BEC

Nithin, Aayush

April 20, 2025

Code for LDPC implementation through BEC

```python
import numpy as np
from itertools import combinations
import random

class LDPCEncoder:
    def __init__(self, k):
        self.k = k
        self.G, self.H = self.construct_generator_and_parity_check()

    def construct_generator_and_parity_check(self):
        subsets = []
        for r in range(self.k, 0, -1):
            subsets.extend(list(combinations(range(self.k), r)))

        n = len(subsets)
        G_full = np.zeros((self.k, n), dtype=int)
        for col_idx, subset in enumerate(subsets):
            for row_idx in subset:
                G_full[row_idx][col_idx] = 1

        P = G_full[:, :-self.k]
        I_k = G_full[:, -self.k:]
        assert np.array_equal(I_k, np.identity(self.k, dtype=int)), "Last k columns of G are not identity."

        G = np.concatenate((P, I_k), axis=1)
        H = np.concatenate((np.identity(n - self.k, dtype=int), P.T), axis=1)

        return G, H

    def encode(self, message):
        return message @ self.G % 2
```

Code 1: LDPC implementation through BEC

Constructing Generator and Parity-check Matrix

- The generator matrix G is constructed from all non-empty subsets of the message bits.
- The matrix is arranged into the form $G = [P \mid I_k]$.
- The parity-check matrix is derived from the above relation as $H = [I_{n-k} \mid P^T]$

Encoding the message

- Once the matrices have been constructed, we can start encoding.

The message is converted to the codeword by mutliplying it by G modulo 2:

$$\text{codeword} = \text{message} \times G \mod 2$$

```python
class BinaryErasureChannel:
    def __init__(self, erasure_prob=0.1):
        self.erasure_prob = erasure_prob

    def transmit(self, codeword):
        received = []
        for bit in codeword:
            if random.random() < self.erasure_prob:
                received.append('e')  # 'e' for erased
            else:
                received.append(str(bit))
        return received
```

Code 2: LDPC implementation through BEC

Simulating the Binary Erasure Channel

- Each bit in the codeword is passed through a simulated channel.
- The bits are replaced with an erasure symbol (`'e'`) with probability ε

```python
class LDPCDecoder:
    def __init__(self, H):
        self.H = H

    def decode(self, received):
        n = len(received)
        codeword = [None if bit == 'e' else int(bit) for bit in received]
        codeword = np.array(codeword, dtype=object)

        max_iterations = 100
        for _ in range(max_iterations):
            updated = False
            for row in self.H:
                indices = [i for i, bit in enumerate(row) if bit == 1]
```

```python
            involved = [codeword[i] for i in indices]

            unknown_indices = [i for i, bit in zip(indices, involved) if bit is None]
            known_values = [bit for bit in involved if bit is not None]

            if len(unknown_indices) == 1:
                known_sum = sum(known_values) % 2
                missing_idx = unknown_indices[0]
                codeword[missing_idx] = known_sum
                updated = True

        if not updated:
            break

    for i in range(len(codeword)):
        if codeword[i] is None:
            print(f" Warning: Bit at index {i} could not be recovered. Defaulting to 0.")
            codeword[i] = 0

    return np.array(codeword, dtype=int)
```

Code 3: LDPC implementation through BEC

Decoding the received message

- The decoder uses H and iterates through each of its rows to get mutliple parity check equations.
- If a parity-check equation has only one unknown bit, it can be solved.
- The iterations take place fixed number of times or until no updates have been made.
- The decoded message is returned in the end.

For example, given message `1011` and $q = 4$, the generated codeword might look like:

$$[1, 0, 1, 0, 1, 1, 0]$$

If bits at position 3 and 6 are erased, the received message would look like:

$$[1, 0, \texttt{'e'}, 0, 1, \texttt{'e'}, 0]$$

And after the desired algorithm we will get it as

$$[1, 0, 1, 0, 1, 1, 0]$$

```python
class CodewordComparator:
    @staticmethod
    def compare(original, decoded):
        return "Success" if np.array_equal(original, decoded) else "Failure"
```

Code 4: LDPC implementation through BEC

Checker function

- Checks if both the encoded codeword and corrected decoded codeword are correct.

```python
# === Main Execution ===
q = int(input("Enter the length of your message chunks (q): \n"))
msg_input = input("Enter the message (as a binary number, e.g., 1011): \n")
epsilon = 0.1  # BEC erasure probability

# Message preprocessing
message = np.array([int(bit) for bit in msg_input.strip()], dtype=int)
assert len(message) == q, f"Message length ({len(message)}) must match q = {q}"

# Instantiate and use classes
encoder = LDPCEncoder(q)
codeword = encoder.encode(message)

bec = BinaryErasureChannel(epsilon)
received = bec.transmit(codeword)

decoder = LDPCDecoder(encoder.H)
decoded_codeword = decoder.decode(received)

# Output
print(f"\n Generator matrix G = [P | I] (shape {encoder.G.shape}):\n{encoder.G}")
print(f"\n Parity-check matrix H = [I | P^T] (shape {encoder.H.shape}):\n{encoder.H}")
print(f"\n Original message: {message}")
print(f" Encoded codeword: {codeword}")
print(f" Received over BEC (={epsilon}): {received}")
print(f"\n Decoded codeword: {decoded_codeword}")
print(CodewordComparator.compare(codeword, decoded_codeword))
```

Code 5: LDPC implementation through BEC

Taking Input

- Input is taken from the user for specifying its length q
- Then the input of the binary message is taken from the user(e.g., `1011`)
- The erasure probability ε is assigned(in this case, 0.1)

- It is then encoded,passed through the channel and then decoded using BEC decoder.

```
Enter the length of your message chunks (q):
4
Enter the message (as a binary number, e.g., 1011):
0100

 Generator matrix G = [P | I] (shape (4, 15)):
[[1 1 1 1 0 1 1 1 0 0 0 1 0 0 0]
 [1 1 1 0 1 1 0 0 1 1 0 0 1 0 0]
 [1 1 0 1 1 0 1 0 1 0 1 0 0 1 0]
 [1 0 1 1 1 0 0 1 0 1 1 0 0 0 1]]

 Parity-check matrix H = [I | P^T] (shape (11, 15)):
[[1 0 0 0 0 0 0 0 0 0 0 1 1 1 1]
 [0 1 0 0 0 0 0 0 0 0 0 1 1 1 0]
 [0 0 1 0 0 0 0 0 0 0 0 1 1 0 1]
 [0 0 0 1 0 0 0 0 0 0 0 1 0 1 1]
 [0 0 0 0 1 0 0 0 0 0 0 0 1 1 1]
 [0 0 0 0 0 1 0 0 0 0 0 1 1 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 1 0 1 0]
 [0 0 0 0 0 0 0 1 0 0 0 1 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 1 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 1]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 1 1]]

 Original message: [0 1 0 0]
 Encoded codeword: [1 1 1 0 1 1 0 0 1 1 0 0 1 0 0]
 Received over BEC (=0.1): ['1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', 'e', '1', '0', '0']

Decoded codeword: [1 1 1 0 1 1 0 0 1 1 0 0 1 0 0]
Success
```

Output 6: An example of the output

References

- David McKay's book on Information Theory | Book |
- Presentation on basic LDPC code encoding and decoding in BSC | Presentation |
- NPTEL Video 1 | Link |
- NPTEL Video 2 | Link |
- NPTEL Video 3 | Link |
- NPTEL Video 4 | Link |
- NPTEL Video 5 | Link |
- Basic Intro Video | |