Turbo Codes Documentation

April 24, 2025

Introduction

Turbo codes are a vital development in error correction techniques, enabling reliable communication over noisy channels. Introduced in 1993 by Claude Berrou, Alain Glavieux, and Punya Thitimajshima, turbo codes revolutionized digital communications by approaching Shannon's theoretical limit of channel capacity. They are based on parallel concatenation of two Recursive Systematic Convolutional (RSC) codes, separated by an interleaver.

Key features of turbo codes include:

- **Iterative Decoding:** Using the soft-in/soft-out BCJR algorithm for decoding.
- **High Error Correction Capability:** Efficiently reduces bit error rate (BER).
- **Flexibility:** Adaptable to a range of systems such as 3G, LTE, and satellite links.

Turbo Coding Process

Encoding

Turbo coding starts with data encoding using two RSC encoders. The original data and its interleaved version are each fed into one encoder. Each encoder produces parity bits. Together with the original bits, they form the transmitted signal.

Interleaving

The interleaver scrambles the order of the input bits before the second encoding step, distributing errors to enhance error correction during decoding.

Puncturing (Not Implemented)

This optional step omits some parity bits to adjust the code rate. It helps in achieving higher data throughput at the cost of slightly reduced error performance.

Implementation Details

This implementation uses Python, TensorFlow, and the Sionna library. TensorFlow enables tensor operations and Sionna provides optimized communication primitives including the BCJR decoder.

Function Descriptions

`gaussian_channel_simulator`

Purpose: Adds Gaussian noise to the modulated signal.
Inputs: Encoded data list, standard deviation of the Gaussian noise.
Output: Noisy encoded signal, simulating a realistic channel.

`zero_padding`

Purpose: Ensures the message length is a multiple of the block size by appending zeros.

`rs_encoder`

Purpose: Applies (7,5) Recursive Systematic Convolutional encoding.
Inputs: Message list, number of information bits in a block, option to enforce final state (0,0).
Output: Encoded message with parity appended.

`interleaver`

Purpose: Applies a permutation based on modular multiplication to scatter bits. But, for this interleaver to work properly, number of information bits in a block must be of the form of a prime - 2, the seed also has some constraints
Inputs: seed(a number), number of information bits in a block + 2, list/array to be permuted.
Output: Interleaved list/array.

`turbo_encoder_without_puncturing`

Purpose: Combines two RSC encoders and an interleaver to generate systematic and redundant bits.

`mix_sys_parity`

Purpose: Alternates between systematic and parity data for combined output.

turbo_decode

Purpose: Iteratively decodes received sequences using the BCJR algorithm imported from sionna library. Here, the seed and its inverse should be such that seed*inverse must give remainder 1 when divided by number of information bits per block+2
Inputs: Systematic, parity1, parity2 sequences; interleaver parameters(seed and its inverse, block length); number of iterations.
Output: Final log-likelihood ratios after iterations.

hard_message_output

Purpose: Converts LLRs into binary hard decisions. Removes final two bits added for (0,0) state.

no_of_bit_errors_finder

Purpose: Computes the number of bit mismatches between original and decoded sequences.

Function Summary

Function Name	Description
gaussian_channel_simulator	Simulates Gaussian noise for a realistic transmission environment.
rs_encoder	Performs RSC encoding adding parity bits and optionally terminating to state (0,0).
interleaver	Permutes the message sequence to enhance error dispersion.
turbo_encoder_without_puncturing	Implements full turbo encoding using two encoders and one interleaver.
turbo_decode	Decodes the encoded data using iterative soft decision decoding (BCJR).
hard_message_output	Converts soft LLRs to binary values for final decisions.
no_of_bit_errors_finder	Calculates total bit errors between source and decoded sequences.

Sample Output

- Encoded message: [1, 0, 1, 0, ..., 0]
- Noisy channel output: [1, 0, 1, ..., 0]
- Padded original message: [1, 0, 1, 0, ..., 0]
- Decoded message: [1, 0, 1, 0, ..., 0]
- Bit errors: 6

- Bit errors without coding: 173

Conclusion

This document presents a detailed guide to implementing turbo codes using Python. With robust components for encoding, interleaving, and decoding, this system provides excellent error correction suitable for modern communication needs.