Polar Codes Python Implementation Documentation

Developer: EE24B069

April 24, 2025

1 Introduction

This document provides a detailed explanation of the Python implementation of Polar Codes as provided in the file. Polar codes are a class of capacity-achieving error-correcting codes. This document explains each function used in the implementation.

2 Function Documentation

2.1 gaussian_channel_simulator

Purpose: Simulates the transmission of a signal over an additive white Gaussian noise (AWGN) channel.

```
def gaussian_channel_simulator(encoded_message_arr, std_dev_of_error):
    modulated_arr = (2 * encoded_message_arr) - 1
    return np.random.normal(0, std_dev_of_error, modulated_arr.shape) + modulated_arr
```

This modulates bits opposite to normal BPSK format, i.e, 0 is mapped to -1 and 1 to mapped to 1, then adds Gaussian noise of given standard deviation.

2.2 worst_to_best_channel_indices_arr_reducer

Purpose: Reduces the worst to best channel indices array to the required number of channels.

```
def worst_to_best_channel_indices_arr_reducer(no_of_channels,
    worst_to_best_channel_indices_arr):
    ...
```

Ensures the returned array has indices only within the range [0, no_of_channels-1].

2.3 channel_inputs_organiser

Purpose: Arranges the input bits and frozen bits before encoding.

```
def channel_inputs_organiser(depth_of_tree, worst_to_best_channel_indices_arr, message_list)
    :
    ...
```

Pads the message with zeros (frozen bits) and reorders them so that the frozen bits go to the least reliable channels.

2.4 polar_operation

Purpose: Combines two lists using bitwise XOR followed by second list as it is.

```
def polar_operation(list1, list2):
    ...
```

Returns the XOR result followed by the second list.

2.5 polar_encoder

Purpose: Encodes a message using the Polar encoding scheme.

```
def polar_encoder(depth_of_tree, ...):
    ...
```

Constructs the encoded message by recursively applying polar operation.

2.6 Node Class

Purpose: Defines a node in the successive cancellation decoding tree. Each node processes log-likelihood ratios (LLRs), and forwards decisions. LLRs have not been exactly used, the received values have been directly used. Includes:

- `minsum_calculater`: Min-sum approximation for the log sum exp, this is the llr's for the left child.
- `llr_for_right_child_calculator`: Computes LLRs for right child, using the hard output from the left node
- `create_and_send_info_to_left/right_child`: Creates a new node and computes their respective LLRs.
- `send_info_above`: Propagates bits upward for further decoding.

2.7 frozen_bits_remover

Purpose: Removes padded zeros(frozen bits) from the decoded message.

```
def frozen_bits_remover(decoded_message_list, ...):
    ...
```

Returns the actual decoded message by excluding frozen bits.

2.8 polar_decoder

Purpose: Decodes a single chunk of received bits.

```
def polar_decoder(received_list, ...):
    ...
```

Implements Successive Cancellation decoding using the Node class.

2.9 complete_polar_encoder and complete_polar_decoder

Purpose: Handle full-length messages by dividing them into chunks. The polar encoder and decoder used before could only handle small message lengths, length of message must be less than number of channels there. Here, it breaks the big message into chunks and overcomes that issue.

```
def complete_polar_encoder(...):
    ...

def complete_polar_decoder(...):
    ...
```

Breaks a long message into multiple parts and processes each chunk separately.

2.10 no_of_bit_errors_finder

Purpose: Calculates bit error count.

```
def no_of_bit_errors_finder(message_list, decoded_message_list):
    ...
```

Compares the decoded message with the original and returns the number of bit errors.

3 Conclusion

This documentation outlines the purpose and working of each component in the polar codes implementation. Together, these functions and classes perform efficient polar encoding and decoding over noisy channels.