TD8, Polar Codes

INF563 Introduction to Information Theory

March 5, 2021

The purpose of this TD is to implement a simple polar code.

1 $(U + V \mid V)$ codes

We will be interested here in binary $(U + V \mid V)$ codes. A binary $(U + V \mid V)$ construction takes two binary codes of a same length n and produces a code of length $2n$ as follows (we denote here the concatenation of two vectors $\mathbf{x}$ and $\mathbf{y}$ by $(\mathbf{x}|\mathbf{y})$)

Definition 1 ($(U + V \mid V)$ binary code) *Let U and V be two binary linear codes of a same length. We define the $(U + V \mid V)$-construction of U and V as the binary linear code:*

$$(U + V \mid V) = \{(\mathbf{u} + \mathbf{v} \mid \mathbf{v}); \mathbf{u} \in U \text{ and } \mathbf{v} \in V\}.$$

The dimension of the $(U + V \mid V)$ code is $k_U + k_V$ and its minimum distance is $\min(2d_V, d_U)$ when the dimensions of U and V are k_U and k_V respectively, the minimum distance of U is d_U and the minimum distance of V is d_V.

A polar code is an iterated $(U + V \mid V)$ code, in the sense that the codes U and V are themselves $(U + V \mid V)$ codes and so and so forth up to the point where the constituent codes are of length 1. Such codes have therefore a length which is a power of 2. For instance a polar code of length 4 is a $(U + V \mid V)$ code of length 4 where U is a code of length 2 which is a $(U + V \mid V)$ construction obtained from two codes of length 1. The same applies to the code V of length 2. A polar code of length 2^n and dimension k is associated in a natural way to a binary tree of depth n with k leaves corresponding to the information bits of the code. It is first asked to show the two properties of a $(U + V \mid V)$ code (about their dimension and their minimum distance). Use the property about the minimum distance and the dimension to find a polar code of length 8, dimension 4 and minimum distance 4.

2 Encoding a polar code

The recursive $(U + V \mid V)$ structure of a polar code leads in a natural way to an encoding algorithm for it. We use the following algorithm to encode a polar code. To encode a polar code of length 2^n and dimension k we take a binary vector $\mathbf{u}$ with $2^n - k$ positions fixed to 0 and k positions where the information bits are copied to. Encoding is performed by calling `Encode`$(0, 2^n)$. The vector $\mathbf{u}$ is changed into its encoded version.

```
function ENCODE(i, j)
    m ← (i + j)/2
    d ← (j − i)/2
    if d > 0 then
        ENCODE(i, m)
        ENCODE(m, j)
        for l = i to m − 1 do
            u[l] ← u[l] ⊕ u[l + d]
```

Implement this function in Java and check that it gives the right encoding of the polar code of length 8 that you have found in the previous question.

3 Decoding a polar code

Assume that we have a decoding algorithm for the U and the V code that uses the fact that we know for each bit i of the codeword the probability p_i that it is equal to 1. Then this can be used to decode a $(U + V \mid V)$ code in the following way again under the assumption that we know for each bit i of the codeword $(\mathbf{u}+\mathbf{v}, \mathbf{v})$ that has been sent the probability that it is equal to 1 given the received symbol y_i for it. We number the positions of the $(U + V \mid V)$ code from 0 to $2n - 1$ and assume that these probabilities are stored in an array $p[0], \ldots, p[2n - 1]$. We view $\mathbf{u}$ and $\mathbf{v}$ as arrays of length n. We assume that the received word is given by the array $y[0], \ldots, y[2n - 1]$. From our assumption, we know that

$$\mathbf{prob}(u[i] + v[i] = 1 | y[i]) = p[i] \tag{1}$$

$$\mathbf{prob}(v[i] = 1 | y[i + n]) = p[i + n] \tag{2}$$

We can use now the lecture on polar codes to deduce that

$$\mathbf{prob}(u[i] = 1 | y[i], y[i + n]) = \frac{1 - (1 - 2p[i])(1 - 2p[i + n])}{2}$$

We can decode U by using these probabilities. Once we know $\mathbf{u}$ we can use the lecture on polar codes to deduce that

$$\mathbf{prob}(v[i] = 1 | y[i], y[i + n], u[i]) = \frac{p[i]p[i + n]}{p[i]p[i + n] + (1 - p[i])(1 - p[i + n])} \text{ if } u[i] = 0$$

$$\mathbf{prob}(v[i] = 1 | y[i], y[i + n], u[i]) = \frac{(1 - p[i])p[i + n]}{(1 - p[i])p[i + n] + p[i](1 - p[i + n])} \text{ if } u[i] = 1.$$

This can be used to decode the $(U + V \mid V)$ code as follows.

function DECODEUV($\mathbf{p}$)

```
for i = 0 to n − 1 do
    q[i] ← (1−(1−2p[i])(1−2p[i+n]))/2
u ← DecodeU(q)
for i = 0 to n − 1 do
    if u[i] = 0 then
        r[i] ← p[i]p[i+n] / (p[i]p[i+n]+(1−p[i])(1−p[i+n]))
    else
        r[i] ← (1−p[i])p[i+n] / ((1−p[i])p[i+n]+p[i](1−p[i+n]))
v ← DecodeV(r)
return (u + v, v)
```

The issue is now: how can we decode U and V ? If U and V were of length 1, then the answer would be easy. For U there are two cases to consider. Either U is the constant code, say $U = \{0\}$ and then `DecodeU` would just return 0 or $U = \{0, 1\}$ and then `DecodeU`$(\mathbf{p})$ would return 0 if $p[0] < \frac{1}{2}$ and 1 otherwise. The recursive $(U + V \mid V)$ structure of a polar code leads in a natural way to a recursive decoding algorithm based on these considerations. One uses here a table $\mathbf{p}[0..n-1][0..2^n-1]$ storing all the probabilities that are computed during the decoding process, where

$$\begin{aligned} n &= \text{number of layers of the polar code} \\ 2^n &= \text{length of the polar code} \\ \mathbf{p}[n][i] &= \mathbf{prob}(x_i = 1|y_i) \\ \mathbf{p}[t][i] &= \text{probability of the } i\text{-th bit at layer } t, \text{ for } t < n \end{aligned}$$

```
function Decode(i, j, t)
    if t = 0  then
        DecodeDirectly(i)
    else
        m ← (i+j)/2
        UpdateU(i, m, t − 1)
        Decode(i, m, t − 1)
        UpdateV(m, j, t − 1)
        Decode(m, j, t − 1,)
        EncodeUV(i, j, t)
```

where the auxiliary functions are defined as

- ```
  function UpdateU(i, j, t)
      for l = i to j − 1 do
          p[t][l] ← (1−(1−2p[t+1][l])(1−2p[t+1][l+2^t]))/2
  ```
- ```
  function UpdateV(i, j, t)
      for ℓ = i to j − 1 do
          p_1 ← p[t + 1][ℓ − 2^t]
          p_2 ← p[t + 1][ℓ]
          if p[t][ℓ − 2^t] = 0 then
  ```

$\mathbf{p}[t][\ell] \leftarrow \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$
else
$\mathbf{p}[t][\ell] \leftarrow \frac{(1-p_1) p_2}{(1-p_1) p_2 + p_1(1-p_2)}$

- **function** EncodeUV(i, j, t)
  for $\ell = i$ to $\frac{i+j}{2} - 1$ **do**
  $\mathbf{p}[t][\ell] \leftarrow \mathbf{p}[t-1][\ell] + \mathbf{p}[t-1][\ell + 2^{t-1}]$
  for $\ell = \frac{i+j}{2}$ to $j - 1$ **do**
  $\mathbf{p}[t][\ell] \leftarrow \mathbf{p}[t-1][\ell]$

- **function** DecodeDirectly(i)
  if $\mathbf{z}[i] \neq 0$ **then**
  $\mathbf{p}[0][i] = \mathbf{z}[i]$
  else
  if $\mathbf{p}[0][i] < \frac{1}{2}$ **then** $\mathbf{p}[0][i] \leftarrow 0$
  else $\mathbf{p}[0][i] \leftarrow 1$

Here $\mathbf{z}$ is a table of length 2^n where the $2^n - k$ positions fixed to 0 at the beginning of the encoding process are also fixed to 0 in $\mathbf{z}$ and the other k positions (where information was fed in during the encoding process) take the value -1. Decoding is performed by using the following function with the call `Decode`$(0, 2^n, n)$.

Implement this function in Java and verify that the decoding is succesful most of the time for the polar code of length 8 that you have found in the previous question when codewords are sent over a binary symmetric channel of crossover probability $p = 0.06$.