CSC310 – Information Theory — Sam Roweis

Lecture 20:

Shannon's Theorm Proof & Product Codes

November 23, 2005

Shannon's Noisy Coding Theorem for the BSC

- Consider a BSC with error probability $f < 1/2$.
  This channel has capacity $C = 1 - H_2(f)$.
- For any desired closeness to capacity, $\eta > 0$, and for any desired limit on error probability, $\epsilon > 0$, there is a code of some length N whose rate, R, is at least $C - \eta$, and for which the probability that nearest neighbor decoding will decode a codeword incorrectly is less than ϵ.
- Last class we started to give a proof of this, which more-or-less follows the proof for general channels in Chapter 10 of MacKay's book.
- The idea is based on showing that a *randomly chosen code* performs quite well and hence that there must be *specific codes* which also perform quite well.

Strategy for Proving the Theorem

- Rather than showing how to construct a specific code for given values of f, η, and ϵ, we will consider choosing a code of a suitable length, N, and rate $\log_2(M)/N$, by picking M codewords *at random* from Z_2^N.
- We consider the following scenario:
  1. We randomly pick a code, $\mathcal{C}$, which we give to both the sender and the receiver.
  2. The sender randomly picks a codeword $\mathbf{x} \in \mathcal{C}$, and transmits it through the channel.
  3. The channel randomly generates an error pattern, $\mathbf{n}$, and delivers $\mathbf{y} = \mathbf{x} + \mathbf{n}$ to the receiver.
  4. The receiver decodes $\mathbf{y}$ to a codeword, $\mathbf{x}^*$, that is nearest to $\mathbf{y}$ in Hamming distance.
- If the probability that this process leads to $\mathbf{x}^* \neq \mathbf{x}$ is $< \epsilon$, then there must be some specific code with error probability $< \epsilon$.

Rearranging the Order of Choices

- It will be convenient to rearrange the order in which random choices are made, as follows:
  1. We randomly pick *one* codeword, $\mathbf{x}$, which is the one the sender transmits.
  2. The channel randomly generates an error pattern, $\mathbf{n}$, that is added to $\mathbf{x}$ to give the received data, $\mathbf{y}$. Let the number of transmission errors (ie, ones in $\mathbf{n}$) be w.
  3. We now randomly pick the other $M-1$ codewords. If the Hamming distance from $\mathbf{y}$ of all these codewords is greater than w, nearest-neighbor decoding will make the correct choice.
- The probability of the decoder making the wrong choice here is the same as before.

Decoding with Typical Error Patterns

- The probability that the codeword nearest to $\mathbf{y}$ is the correct decoding will be at least as great as the probability that the following sub-optimal decoder decodes correctly:

  If there is exactly one codeword $\mathbf{x}^*$ for which $\mathbf{n} = \mathbf{y} - \mathbf{x}^*$ has a typical number of ones, then decode to $\mathbf{x}^*$, otherwise declare that decoding has failed.

- This sub-optimal decoder can fail in two ways:
  - The correct decoding, $\mathbf{x}$, may correspond to an error pattern, $\mathbf{n} = \mathbf{y} - \mathbf{x}$, that is not typical.
  - Some other codeword, $\mathbf{x}'$, may exist for which the error pattern $\mathbf{n}' = \mathbf{y} - \mathbf{x}'$ is typical.

Bounding the Probability of Failure (I)

- The total probability of decoding failure is less than the sum of the probabilities of failing in these two ways.
  We will try to limit each of these to $\epsilon/2$.
- We can choose N to be big enough that

$$P(f - \beta < w/N < f + \beta) \;\geq\; 1 - \epsilon/2$$

  This ensures that the actual error pattern will be non-typical with probability less than $\epsilon/2$.
- We now need to limit the probability that some other codeword also corresponds to a typical error pattern.

Bounding the Probability of Failure (II)

- The number of typical error patterns is

$$J \;<\; 2^{N(H_2(f)+\beta\log_2((1-f)/f))}$$

- For a random codeword, $\mathbf{x}$, other than the one actually transmitted, the corresponding error pattern given $\mathbf{y}$ will contain 0s and 1s that are independent and equally likely.
- The probability that one such codeword will produce a typical error pattern is therefore

$$J/2^N \;<\; 2^{-N(1-H_2(f)-\beta\log_2((1-f)/f))}$$

- The probability that *any* of the other $M-1$ codewords will correspond to a typical error pattern is bounded by M times this. We need this to be less than $\epsilon/2$, ie

$$M\,2^{-N(1-H_2(f)-\beta\log_2((1-f)/f))} \;<\; \epsilon/2$$

Finishing the Proof

- Finally, we need to pick β, M, and N so that the two types of error have probabilities less than $\epsilon/2$, and the rate, R is at least $C - \eta$.
- We will let $M = 2^{\lceil (C-\eta)N \rceil}$, and make sure N is large enough that $R = \lceil (C-\eta)N \rceil / N < C$.
- With this value of M, we need

$$2^{\lceil (C-\eta)N \rceil}\; 2^{-N(1-H_2(f)-\beta\log_2((1-f)/f))} < \epsilon/2$$
$$\Rightarrow 2^{-N(1-H_2(f)-\lceil (C-\eta)N \rceil/N-\beta\log_2((1-f)/f))} < \epsilon/2$$

- The channel capacity is $C = 1 - H_2(f)$, so that $1 - H_2(f) - \lceil (C-\eta)N \rceil / N \;=\; C - R$ is positive.
- For a sufficiently small value of β, $1 - H_2(f) - \lceil (C-\eta)N \rceil / N - \beta\log_2((1-f)/f)$ will also be positive. With this β and a large enough N, the probabilities of both types of error will be less than $\epsilon/2$, so the total error probability will be less than ϵ.

Good Codes and Minimum Distance

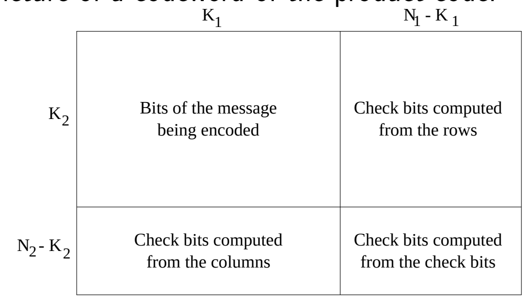
- Recall that for a code to be guaranteed to correct up to t errors, it's minimum distance must be at least $2t + 1$.
- What's the minimum distance for the random codes used to prove the noisy coding theorem?
- A random N-bit code is very likely to have minimum distance $d \leq N/2$ — if we pick two codewords randomly, about half their bits will differ. So these codes are likely *not guaranteed* to correct patterns of $N/4$ or more errors.
- A BSC with error probability f will produce about Nf errors. So for $f > 1/4$, we expect to get more errors than the code is guaranteed to correct. Yet we know these codes are good!
- **Conclusion:** A code may be able to correct *almost all* patterns of t errors even if it can't correct *all* such patterns.

Product Codes

- A *product code* is formed from two other codes $\mathcal{C}_1$, of length N_1, and $\mathcal{C}_2$, of length N_2. The product code has length $N_1 N_2$.
- We can visualize the $N_1 N_2$ symbols of the product code as a 2D array with N_1 columns and N_2 rows.
- Definition of a product code: An array is a codeword of the product code if and only if
  - all its rows are codewords of $\mathcal{C}_1$
  - all its columns are codewords of $\mathcal{C}_2$
- We will assume here that $\mathcal{C}_1$ and $\mathcal{C}_2$ are linear codes, in which case the product code is also linear. (Can you see why?)

Dimensionality of Product Codes

- Suppose $\mathcal{C}_1$ is an $[N_1, K_1]$ code and $\mathcal{C}_2$ is an $[N_2, K_2]$ code. Then their product will be an $[N_1 N_2, K_1 K_2]$ code.
- Suppose $\mathcal{C}_1$ and $\mathcal{C}_2$ are in systematic form. Here's a picture of a codeword of the product code:



- The dimensionality of the product code is not more than $K_1 K_2$, since the message bits in the upper-left determine the check bits.
- We'll see that the dimensionality equals $K_1 K_2$ by showing how to find correct check bits for any message.

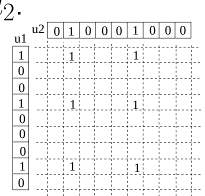
Encoding Product Codes

- Here's a procedure for encoding messages with a product code:
  1. Put $K_1 K_2$ message bits into the upper-left K_2 by K_1 corner of the N_2 by N_1 array.
  2. Compute the check bits for the first K_2 rows, according to $\mathcal{C}_1$.
  3. Compute the check bits for the N_1 columns, according to $\mathcal{C}_2$.
- After this, all the columns will be codewords of $\mathcal{C}_2$, since they were given the right check bits in step (3). The first K_2 rows will be codewords of $\mathcal{C}_1$, since they were given the right check bits in step (2). But are the *last* $N_2 - K_2$ rows codewords of $\mathcal{C}_1$?
- Yes! Check bits are linear combinations of message bits. So the last $N_2 - K_2$ rows are linear combinations of earlier rows. Since these rows are in $\mathcal{C}_1$, their combinations are too.

Minimum Distance of Product Codes

- If $\mathcal{C}_1$ has minimum distance d_1 and $\mathcal{C}_2$ has minimum distance d_2, then the minimum distance of their product is $d_1 d_2$.
- **Proof:**
  Let $\mathbf{u}_1$ be a codeword of $\mathcal{C}_1$ of weight d_1 and $\mathbf{u}_2$ be a codeword of $\mathcal{C}_2$ of weight d_2. Build a codeword of the product code by putting $\mathbf{u}_1$ in row i of the array if $\mathbf{u}_2$ has a 1 in position i. Put zeros elsewhere. This codeword has weight $d_1 d_2$.

  The new codeword is the outer product of the vectors $\mathbf{u}_1$ and $\mathbf{u}_2$.



- Furthermore, any non-zero codeword must have at least this weight. It must have at least d_2 rows that aren't all zero, and each such row must have at least d_1 ones in it.

Decoding Product Codes

- Products of even small codes (eg, $[7, 4]$ Hamming codes) have lots of check bits, so decoding by building a syndrome table may be infeasible.
- But if $\mathcal{C}_1$ and $\mathcal{C}_2$ can easily be decoded, we can decode the product code by first decoding the rows (replacing them with the decoding), then decoding the columns. (Or the other way around.)
- This will usually **not** be a nearest-neighbor decoder (and hence will be sub-optimal, assuming a BSC and equally-likely messages).

How Good are Products of Codes?

- Let $\mathcal{C}$ be an $[N, K]$ code of minimum distance d (guaranteed to correct $t = \lfloor (d-1)/2 \rfloor$ errors).
- How good is the code obtained by taking the product of $\mathcal{C}$ with itself p times?
  - Length: $N_p = N^p$
  - Rate: $R_p = K^p / N^p = (K/N)^p \rightarrow 0$
  - Distance: $d_p = d^p$
  - Relative distance: $\rho_p = d_p / N_p = (d/N)^p \rightarrow 0$
- The code can correct up to about $d_p/2$ errors, corresponding to a proportion of errors of $\rho_p/2$.
- For a BSC with error probability f, we expect that for large N, the proportion of erroneous bits in a block will be very close to f. (The Law of Large Numbers once again.)

Why use Products of Codes?

- The analysis above shows that for large N, these product codes are both *unlikely* to correct all errors, and also that they have a low rate (approaching zero)!
- So why would we ever use them?
- One advantage of product codes: They can correct some *burst errors* — errors that come together, rather than independently.

0	1	?	0	0	1	0	0	0	?	?	?	?	?	?	0	0	0	0	1	0	0	0	1	?	0	0