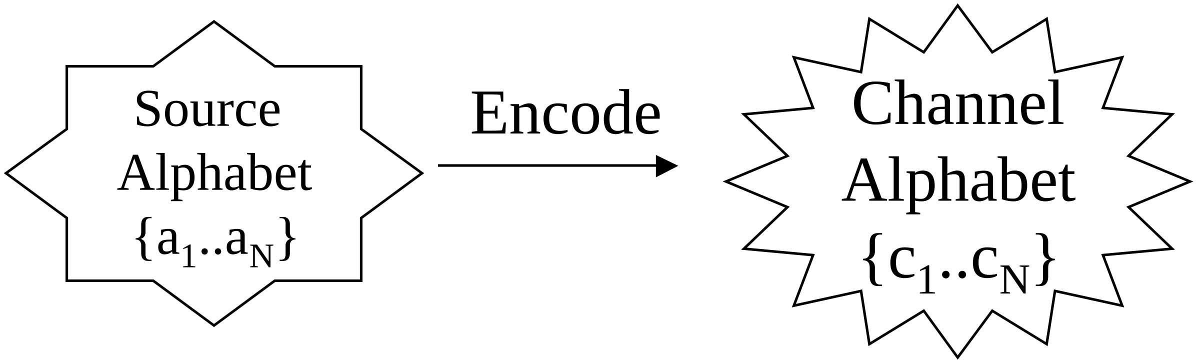
16.36 Communication Systems Engineering
Spring 2009

16.36: Communication Systems Engineering

Lecture 5: Source Coding

Eytan Modiano

Source coding



- **Source symbols**
  - **Letters of alphabet, ASCII symbols, English dictionary, etc...**
  - **Quantized voice**
- **Channel symbols**
  - **In general can have an arbitrary number of channel symbols**
    - **Typically {0,1} for a binary channel**
- **Objectives of source coding**
  - **Unique decodability**
  - **Compression**
    - **Encode the alphabet using the smallest average number of channel symbols**

Compression

- **Lossless compression**
  - **Enables error free decoding**
  - **Unique decodability without ambiguity**

- **Lossy compression**
  - **Code may not be uniquely decodable, but with very high probability can be decoded correctly**

Prefix (free) codes

- A prefix code is a code in which no codeword is a prefix of any other codeword
  - Prefix codes are uniquely decodable
  - Prefix codes are instantaneously decodable

- The following important inequality applies to prefix codes and in general to all uniquely decodable codes

Kraft Inequality

Let $n_1 \ldots n_k$ be the lengths of codewords in a prefix (or any Uniquely decodable) code. Then,

$$\sum_{i=1}^{k} 2^{-n_i} \leq 1$$

Proof of Kraft Inequality

- **Proof only for prefix codes**
  - **Can be extended for all uniquely decodable codes**

- **Map codewords onto a binary tree**
  - **Codewords must be leaves on the tree**
  - **A codeword of length n_i is a leaf at depth n_i**

- **Let $n_k \geq n_{k-1} \ldots \geq n_1 \Rightarrow$ depth of tree = n_k**
  - **In a binary tree of depth n_k, up to 2^{nk} leaves are possible (if all leaves are at depth n_k)**
  - **Each leaf at depth $n_i < n_k$ eliminates a fraction $1/2^{ni}$ of the leaves at depth $n_k \Rightarrow$ eliminates 2^{nk-ni} of the leaves at depth n_k**
  - **Hence,**

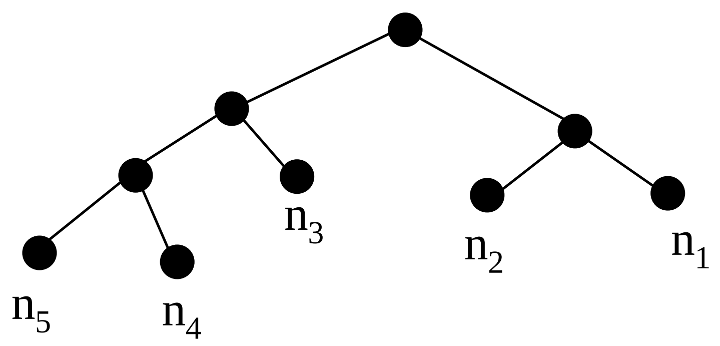
$$\sum_{i=1}^{k} 2^{n_k - n_i} \leq 2^{n_k} \Rightarrow \sum_{i=1}^{k} 2^{-n_i} \leq 1$$

Kraft Inequality - converse

- If a set of integers $\{n_1..n_k\}$ satisfies the Kraft inequality the a prefix code can be found with codeword lengths $\{n_1..n_k\}$
  - Hence the Kraft inequality is a necessary and sufficient condition for the existence of a uniquely decodable code

- Proof is by construction of a code
  - Given $\{n_1..n_k\}$, starting with n_1 assign node at level n_i for codeword of length n_i. Kraft inequality guarantees that assignment can be made

<u>Example:</u> n = {2,2,2,3,3}, (verify that Kraft inequality holds!)



Average codeword length

- Kraft inequality does not tell us anything about the average length of a codeword. The following theorem gives a tight lower bound

Theorem: Given a source with alphabet $\{a_1..a_k\}$, probabilities $\{p_1..p_k\}$, and entropy H(X), the average length of a uniquely decodable binary code satisfies:

$$\bar{n} \geq H(X)$$

Proof:

$$H(X) - \bar{n} = \sum_{i=1}^{i=k} p_i \log \frac{1}{p_i} - \sum_{i=1}^{i=k} p_i n_i = \sum_{i=1}^{i=k} p_i \log \frac{2^{-n_i}}{p_i}$$

$$\log inequality \Rightarrow \log(X) \leq X - 1 \Rightarrow$$

$$H(X) - \bar{n} \leq \sum_{i=1}^{i=k} p_i \left[\frac{2^{-n_i}}{p_i} - 1 \right] = \sum_{i=1}^{i=k} 2^{-n_i} - 1 \leq 0$$

Average codeword length

- **Can we construct codes that come close to H(X)?**

Theorem: Given a source with alphabet $\{a_1..a_k\}$, probabilities $\{p_1..p_k\}$, and entropy H(X), it is possible to construct a prefix (hence uniquely decodable) code of average length satisfying:

$$\bar{n} < H(X) + 1$$

Proof (Shannon-fano codes):

$$Let\ \mathbf{n_i} = \left\lceil \log(\frac{1}{p_i}) \right\rceil \Rightarrow \mathbf{n_i} \geq \log(\frac{1}{p_i}) \Rightarrow 2^{-\mathbf{n_i}} \leq p_i$$

$$\Rightarrow \sum_{i=1}^{k} 2^{-\mathbf{n_i}} \leq \sum_{i=1}^{k} p_i \leq 1$$

$\Rightarrow$ **Kraft inequality satisfied!**

$\Rightarrow$ **Can find a prefix code with lengths,**

$$\mathbf{n_i} = \left\lceil \log(\frac{1}{p_i}) \right\rceil < \log(\frac{1}{p_i}) + 1$$

$$\mathbf{n_i} = \left\lceil \log(\frac{1}{p_i}) \right\rceil < \log(\frac{1}{p_i}) + 1,$$

Now,

$$\bar{n} = \sum_{i=1}^{k} p_i n_i < \sum_{i=1}^{k} p_i \left[\log(\frac{1}{p_i}) + 1 \right] = H(X) + 1.$$

Hence,

$$H(X) \leq \bar{n} < H(X) + 1$$

Getting Closer to H(X)

- **Consider blocks of N source letters**
  - **There are K^N possible N letter blocks (N-tuples)**
  - **Let Y be the "new" source alphabet of N letter blocks**
  - **If each of the letters is independently generated,**

$$H(Y) = H(x_1..x_N) = N^*H(X)$$

- **Encode Y using the same procedure as before to obtain,**

$$H(Y) \le \bar{n}_y < H(Y) + 1$$
$$\Rightarrow N * H(X) \le \bar{n}_y < N * H(X) + 1$$
$$\Rightarrow H(X) \le \bar{n} < H(X) + 1/N$$

Where the last inequality is obtained because each letter of Y corresponds to N letters of the original source

- **We can now take the block length (N) to be arbitrarily large and get arbitrarily close to H(X)**

Huffman codes

- **Huffman codes are special prefix codes that can be shown to be optimal (minimize average codeword length)**

<u>Huffman Algorithm:</u>

1) Arrange source letters in decreasing order of probability ($p_1 \geq p_2 .. \geq p_k$)

2) Assign '0' to the last digit of X_k and '1' to the last digit of X_{k-1}

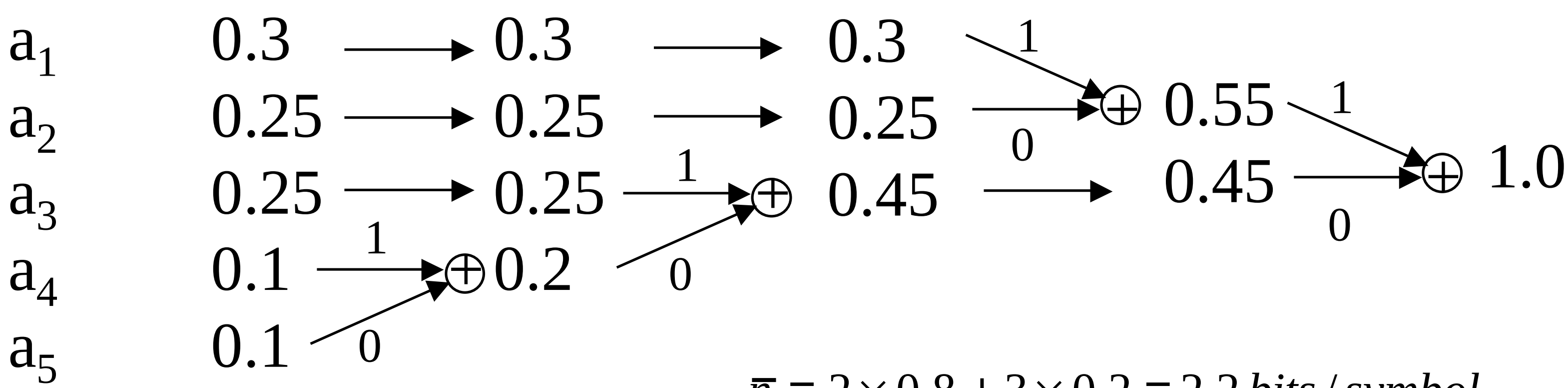
3) Combine pk and pk-1 to form a new set of probabilities

$$\{p_1, p_2 ,.., p_{k-2},(p_{k-1}+ p_k)\}$$

4) If left with just one letter then done, otherwise go to step 1 and repeat

Huffman code example

A = {a_1,a_2,a_3, a_4, a_5} and p = {0.3, 0.25,0.25, 0.1, 0.1}



$$\bar{n} = 2\times 0.8 + 3\times 0.2 = 2.2\ bits/symbol$$

Letter	Codeword
a_1	11
a_2	10
a_3	01
a_4	001
a_5	000

$$H(X) = \sum p_i \log(\frac{1}{p_i}) = 2.1855$$

$$Shannon - Fano\ codes \Rightarrow n_i = \left\lceil \log(\frac{1}{p_i}) \right\rceil$$

$$n_1 = n_2 = n_3 = 2,\ \ n_4 = n_5 = 4$$

$$\Rightarrow \bar{n} = 2.4\ bits/symbol < H(X) + 1$$

Lempel-Ziv Source coding

- **Source statistics are often not known**
- **Most sources are not independent**
  - **Letters of alphabet are highly correlated**
    - **E.g., E often follows I, H often follows G, etc.**
- **One can code "blocks" of letters, but that would require a very large and complex code**
- **Lempel-Ziv Algorithm**
  - **"Universal code" - works without knowledge of source statistics**
  - **Parse input file into unique phrases**
  - **Encode phrases using fixed length codewords**
    - **Variable to fixed length encoding**

Lempel-Ziv Algorithm

- **Parse input file into phrases that have not yet appeared**
  - **Input phrases into a dictionary**
  - **Number their location**

- **Notice that each new phrase must be an older phrase followed by a '0' or a '1'**
  - **Can encode the new phrase using the dictionary location of the previous phrase followed by the '0' or '1'**

Lempel-Ziv Example

Input: 0010110111000101011110

Parsed phrases: 0, 01, 011, 0111, 00, 010, 1, 01111

Dictionary

Loc	binary rep	phrase	Codeword	comment
0	0000	null		
1	0001	0	0000 0	loc-0 + ‘0’
2	0010	01	0001 1	loc-1 + ‘1’
3	0011	011	0010 1	loc-2 + ‘1’
4	0100	0111	0011 1	loc-3 + ‘1’
5	0101	00	0001 0	loc-1 +’0’
6	0110	010	0010 0	loc-2 + ‘0’
7	0111	1	0000 1	loc-0 + ‘1’
8	1000	01111	0100 1	loc-4 + ‘1’

Sent sequence: 00000 00011 00101 00111 00010 00100 00001 01001

Notes about Lempel-Ziv

- **Decoder can uniquely decode the sent sequence**
- **Algorithm clearly inefficient for short sequences (input data)**
- **Code rate approaches the source entropy for large sequences**
- **Dictionary size must be chosen in advance so that the length of the codeword can be established**
- **Lempel-Ziv is widely used for encoding binary/text files**
  - **Compress/uncompress under unix**
  - **Similar compression software for PCs and MACs**