1 Some Intuitions About Entropy of Deterministic Functions of Random Variables

Let $Y = f(X)$ where $f(X)$ is deterministic function, and $X$ is a random variable. It is easy to see that

$$H(Y) \leq H(X).$$

This can be intuitively explained as follows.

$$P(Y = y) = \sum_{x_i | f(x_i) = y} P(X = x_i).$$

In the case $f(.)$ is a one-to-one function and therefore inverse $f^{-1}(.)$ exists, $H(Y) = H(X)$ since $P(Y = y) = P(X = f^{-1}(y))$. Since $p(x_i) = p(y_i)$, $H(X) = H(Y)$. In the case when there are more than one value of $X$ that can be mapped to a value of $Y$, then there is less random in $Y$ compared to $X$. Hence, $H(Y) < H(X)$.

Also, $H(X, Y) = H(X)$. This is because $H(X, Y) = H(X) + H(Y|X)$, and the uncertainty of $H(Y|X)$ is 0 since knowing $X$ once know $Y$ immediately. On the other hand, $H(X, Y)$ can be larger than $H(Y)$ since $H(X, Y) = H(Y) + H(X|Y)$, and $H(X|Y)$ can be non-zero, i.e., knowing $Y$ one might not know $X$ precisely since there are many values of $X$ that can map to $Y$.

2 Symbol Code

**Theorem 2.1** (Converse of Kraft inequality) If $\sum_{i=1}^{|X|} D^{-l_i} \leq 1$, then there exists a prefix code with codeword lengths $l_1, l_2, \ldots, l_{|X|}$.
**Proof** Assume $l_i \leq l_{i+1}$, and think of a codeword as a $D$-based expansion: $0.d_1d_2\ldots d_l$. Construct the codeword $C_k = \sum_{i=1}^{k-1} D^{-l_i}$ with $l_k$ digits. Now, for any $j < k$, we have:

$$C_k = C_j + \sum_{i=1}^{k-1} D^{-l_i} \geq C_j + D^{-l_j}. \quad (1)$$

The Eq. implies that $C_j$ cannot be a prefix of $C_k$ because they differ in first $l_j$ digits. For example, let $C_j = 0.0101$, then $C_k \geq 0.0101 + 2^{-4} = 0.0101 + 0.0001 = 0.0110$. Obviously, $C_j$ is different from $C_k$ in the first 4 digits.

**Example 2.2** In this example, we will construct a prefix code based on the given codewords length. Let $l = [2; 2; 3; 3; 3]$. Let’s check Kraft inequality:

$$\sum_{i=1}^{5} 2^{-l_i} = 2^{-2} + 2^{-2} + 2^{-3} + 2^{-3} + 2^{-3} = 0.875 \leq 1.$$ 

Hence, the Kraft’s inequality satisfies.

Following the previous constructive proof, we have the following code:

<table>
<thead>
<tr>
<th>$l_k$</th>
<th>$C_k = \sum_{i=1}^{k-1} D^{-l_i}$</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0 = 0.00</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>0.25 = 0.01</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.5 = 0.100</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>0.625 = 0.101</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>0.75 = 0.110</td>
<td>110</td>
</tr>
</tbody>
</table>

**3 Optimal Code**

We all know that in English, the word *the* occurs more often than the word *magic*. Suppose we want to translate an English paragraph into bits 0 and 1 that results in as few number of bits as possible. To do so, intuitively, we should use fewer bits for *the* than *magic*. This shows that code depends on the pmf on the occurrences of different English words. In addition, for a particular paragraph, it might happen that the word *magic* occurs more than the word *the*. In that case, using a coding scheme that uses fewer bits for *the* than *magic* is not effective. However, we assume that on average, the word *the* will occur more often than the word *magic*. Thus, the optimal code aims to minimize the expected code length over all paragraphs rather than a particular one. That said, we define the optimal code in the definition below.

**Definition 3.1** (Optimal Code): Let $C$ be a uniquely decodable code. Define $l(C(x))$ as the length of the codeword $C(x)$, then $C$ is optimal if $L_C \triangleq E[l(C(x))]$ is as small as possible for a given $p(x)$. Furthermore, we have:

$$L_C \geq H(X)/\log_2 D,$$
where \( D \) is the size of the destination alphabet.

**Proof** Define \( q \) by

\[
q(x) = \frac{D^{-l(x)}}{\sum_x D^{-l(x)}}.
\]

Let \( K = \sum_x D^{-l(x)} \) Note that \( q(x) \) is a valid pmf (probability mass function).

\[
L_c - \frac{H(X)}{\log_2 D} = E[l(C(X))] - \frac{E[\log_2 p(X)]}{\log_2 D}
= E[-\log_D D^{-l(x)}] + E[\log_D p(X)]
= E[-\log_D q(X)K + \log_D p(X)]
= E[\log_D \frac{p(X)}{q(X)}] - \log_D K
= \log_D 2[D(p||q) - \log K] \quad \text{(Convert from log}_D \text{ to log}_2)
\]

(Note: \( \log_B A = \frac{\log_D A}{\log_D B} \))

\[
\log_D 2, D(p||q) \geq 0, \log K \leq 0, \quad (K = \sum_x D^{-l(x)} \leq 1 \text{ from Kraft inequality})
\Rightarrow \log_D 2[D(p||q) - \log K] \geq 0.
\]

### 3.1 Summary

- **Symbol code**
  - Non-singular codes: \( x_1 \neq x_2 \Rightarrow C(x_1) \neq C(x_2) \)
  - Uniquely decodable codes: \( C^+ \) is non-singular, that is \( C^+(x^+) \) is unambiguous.
  - Prefix codes (instantaneous codes): No codeword is a prefix of another
    - Prefix \( \Rightarrow \) Uniquely decodable \( \Rightarrow \) Non-singular

- **Kraft inequality for uniquely decodable code**
  \[ \sum_x D^{-l(x)} \leq 1 \]

- **Lower bound for any uniquely decodable code**
  \[ L_C \geq \frac{H(X)}{\log_2 D} \]
4 Practical Symbol Codes

- Fano code (not optimal)
- Shannon code (not optimal)
- Huffman code (optimal)

4.1 Fano Code

Put the probabilities in decreasing order
Split as close to 50-50 as possible. Repeat with each half

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.20</td>
<td>0</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0.19</td>
<td>1</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0.17</td>
<td>1</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>0.15</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0.14</td>
<td>1</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>0.06</td>
<td>1</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0.05</td>
<td>0</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>0.04</td>
<td>1</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>

$H(p) = 2.81$ bits. $L_F = 2.89$ bits = $0.2 \times 2 + 0.19 \times 3 + 0.17 \times 3 + 0.15 \times 3 + 0.14 \times 3 + 0.06 \times 3 + 0.05 \times 4 + 0.04 \times 4$. It is not optimal! $L_{opt} = 2.85$ bits.

4.2 Conditions for Optimal Prefix Code (Assuming binary prefix code)

An optimal prefix code must satisfy:

- $p(x_i) > p(x_j) \Rightarrow l(x_i) \leq l(x_j)$ (else swap them)
- The two longest codeword must have the same length (else chop a bit off the codeword)
- In the tree corresponding to the optimum code, there must be two branches stemming from each intermediate node
4.3 Huffman Code Construction

1. Take the two smallest $p(x_i)$ and assign each a different last bit. Then merge into a single symbol.
2. Repeat step 1 until only one symbol remains.

**Example 4.1** Fig. 4.3 shows an example of Huffman code construction.

$$C(a) = 00, C(b) = 10, C(c) = 11, C(d) = 010, C(e) = 011.$$  

*Note that the decoding is simple. The decoder just looks at the bits one at a time from left to right. Depending on the value of the bits, it traverses the upper or lower branches to get to the leaves. It then outputs the values at the leaves.*

**Example 4.2**

<table>
<thead>
<tr>
<th>Letter</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.4</td>
<td>00</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.2</td>
<td>01</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.2</td>
<td>10</td>
</tr>
<tr>
<td>$x_4$</td>
<td>0.1</td>
<td>110</td>
</tr>
<tr>
<td>$x_5$</td>
<td>0.1</td>
<td>111</td>
</tr>
</tbody>
</table>

4.3.1 Optimality of Huffman Code

Fig. 3 illustrates the Huffman codes for different levels. $c_i$ is the huffman code for the distribution $p_i$. Specifically,

- $p_2 = [0.55, 0.45], C_2 = [0 1], L_2 = 1$
- $p_3 = [0.25, 0.45, 0.3], C_3 = [00 1 01], L_3 = 1.55$
- $p_4 = [0.25, 0.25, 0.2, 0.3], C_4 = [00 10 11 01], L_4 = 2$
- $p_5 = [0.25, 0.25, 0.2, 0.15, 0.15], C_5 = [00 10 11 010 011], L_5 = 2.3$

We will show that all of these codes (include $C_5$) is optimal. The proof is by contradiction.
First, suppose that $\exists m > 0$ with $C_m$ is the first suboptimal code. Note that $C_2$ is optimal since there are only two symbols so one has to assign 0 and 1 to two different symbols. Second, let $C_m$ be an optimal code, and hence $L_{C_m} < L_{C_m'}$.

Now, rearrange the symbol with the longest codes in $C_m'$ so that the two lowest probabilities $p_i$ and $p_j$ differ only in the last digit. This does not change the optimality since both symbols have the same length. Next, merge $x_i$ and $x_j$ to create a new code.

$$L_{C_m'-1} = L_{C_m'} - (p_i + p_j).$$

This is because the expected code length is reduced by $(p_i + p_j)$ after merging. Specifically, if one replaces $q = p_i + p_j$, then

$$L_{C_m'-1} = \sum_{k \notin \{i,j\}} p_k l_k + q(l_i - 1) = \sum_{k \notin \{i,j\}} p_k l_k + (p_i + p_j)(l_i - 1) = \sum_k p_k l_k - p_i - p_j = L_{C_m'} - (p_i + p_j)$$

But now, $L_{C_m-1} = L_{C_m} - p_i - p_j$.

Since $L_{C_m'} < L_{C_m}$ by assumption, therefore $L_{C_m'-1} < L_{C_m-1}$, but this contradicts the assumption that $C_m$ is the first suboptimal. Therefore, Huffman construction must be optimal at every level.

### 4.4 Optimal Code from Optimization Perspective

If $l(x) = \text{length}(C(x))$ then $C$ is optimal if $L_C = E[l(X)]$ is as small as possible. We want to minimize

$$E[l(X)] = \sum_i p(x_i) l(x_i)$$
Subject to:

\[ \sum_i D^{-l(x_i)} \leq 1 \quad (Kraft's \ inequality) \]

and

\[ l(x_i) \text{ are integers} \]

The optimization problem above is hard due to the integer constraints. However, we can relax the integer constraints. As a result, we do not find the smallest code length, but a lower bound. We reformulate the problem as follows:

Minimize \[ \sum_i p(x_i)l(x_i) \]
Subject to: \[ \sum_i D^{-l(x_i)} \leq 1 \]

Use lagrange multiplier method:

If one wants to minimize \( f(x) \) subject to \( h(x) = 0 \), then you need to find \( x \) such that

\[ \frac{d(f(x) + \lambda h(x))}{dx} = 0 \quad (1) \]

1. Simplify the problem by letting \( l(x) \) to be non-integer, and \( \sum_i D^{-l(x_i)} = 1 \)
2. Substitute in (1) we have:

\[ \frac{d(\sum p_i l_i) + \lambda(\sum D^{-l_i} - 1)}{dl_i} = 0 \]

\[ p_i - \lambda D^{-l_i} \ln D = 0 \quad (2) \]

\[ D^{-l_i} = \frac{p_i}{\ln D}, \forall i. \text{ Sum over all } i, \]

\[ \sum p_i = \lambda \ln D \sum P^{-l_i} \]

\[ \lambda = \frac{1}{\ln D}. \text{ Subtract } \lambda \text{ into (2)}, \]

\[ p_i - \frac{1}{\ln D} D^{l_i} \ln D = 0 \]

\[ p_i = D^{-l_i}, \quad \Rightarrow l_i = -\log_D p_i \]

\[ E[l(x)] = -\sum p_i \log_D p_i = H(x)/\log_2 D. \]

4.5 Shannon code

\[ l_i = \log_2 1/p_i : \quad \hat{l}_i = \lceil \log_2 1/p_i \rceil \]