Lecture 6: Data Compression
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes

Dr. Nhi Tran (ECE-University of Akron)
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Definition (Entropy Rate)

The *entropy rate* of a stochastic process \( \{X_i\} \) is defined by

\[
H(X) = \lim_{n \to \infty} \frac{1}{n} H(X_1, \ldots, X_n)
\]

Definition (Alternative Notation)

\[
H'(X) = \lim_{n \to \infty} H(X_n | X_{n-1} \ldots, X_1)
\]

- \( H(X) \): per symbol entropy of \( n \) RVs.
- \( H'(X) \): Conditional entropy of the last RV given the past.
Stochastic Process with Discrete RVs

All definitions above apply to continuous RVs. For discrete ones, one can use PMF instead of PDF.

**Definition**

A discrete-time stochastic process \( \{X_i\}, i \in \mathcal{I} \) is one for which we associate the discrete index set \( \mathcal{I} = 1, 2, \ldots \) with time.

**Definition**

A discrete-time stochastic process is said to be stationary if the joint PMF of any subset of the sequence of random variables is invariant with respect to shifts in the time index; that is,

\[
\Pr (X_1 = x_1, \ldots, X_n = x_n) = \Pr (X_{1+l} = x_1, \ldots, X_{n+l} = x_n)
\]

for every \( n \) and every shift \( l \) and for all \( x_1, \ldots, x_n \) in \( \mathcal{X} \).
Markov Process or Markov Chain

A simple example of stochastic process with dependence: Each RV depends only the one preceding it and conditionally independent of all other preceding RVs. Such process is said to be Markov

**Definition**

A discrete-time stochastic process $X_1, X_2, \ldots$ is said to be Markov chain or Markov process if for $n = 1, 2, \ldots$

$$\Pr (X_{n+1} = x_{n+1} | X_n = x_n, \ldots, X_1 = x_1) = \Pr (X_{n+1} = x_{n+1} | X_n = x_n)$$

for all $x_1, x_2, \ldots, x_n, x_{n+1} \in \mathcal{X}$.

In this case, the joint PMF is

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2)\ldots p(x_n|x_{n-1})$$
Markov Process or Markov Chain

A simple example of stochastic process with dependence: Each RV depends only the one preceding it and conditionally independent of all other preceding RVs. Such process is said to be Markov

Definition

A discrete-time stochastic process $X_1, X_2, \ldots$ is said to be Markov chain or Markov process if for $n = 1, 2, \ldots$

$$
\Pr (X_{n+1} = x_{n+1} | X_n = x_n, \ldots, X_1 = x_1) = \Pr (X_{n+1} = x_{n+1} | X_n = x_n)
$$

for all $x_1, x_2, \ldots, x_n, x_{n+1} \in \mathcal{X}$.

In this case, the joint PMF is

$$
p(x_1, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2) \ldots p(x_n|x_{n-1})
$$
Markov Process or Markov Chain

A simple example of stochastic process with dependence: Each RV depends only the one preceding it and conditionally independent of all other preceding RVs. Such process is said to be Markov.

**Definition**

A discrete-time stochastic process $X_1, X_2, \ldots$ is said to be *Markov chain* or *Markov process* if for $n = 1, 2, \ldots$

$$\Pr(X_{n+1} = x_{n+1} | X_n = x_n, \ldots, X_1 = x_1) = \Pr(X_{n+1} = x_{n+1} | X_n = x_n)$$

for all $x_1, x_2, \ldots, x_n, x_{n+1} \in \mathcal{X}$.

In this case, the joint PMF is

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2)\ldots p(x_n|x_{n-1})$$
Entropy Rate of A Stationary Stochastic Process

**Theorem**

For a stationary stochastic process, $H(X)$ and $H'(X)$ exist and are equal:

For a stationary Markov chain, one has:

$$H(X) = H'(X) = \lim_{n \to \infty} H(X_n | X_{n-1} \ldots, X_1) = \lim_{n \to \infty} H(X_n | X_{n-1})$$

$$= H(X_2 | X_1)$$

where the initial state is drawn according to a stationary distribution $\mu$, i.e., $\mu_j = \sum_i \mu_i P_{ij} \forall j$. Can be more explicit?

**Theorem**

$$H(X) = H(X_2 | X_1) = \sum_i \mu_i \left( \sum_j -P_{ij} \log P_{ij} \right) = -\sum_{ij} \mu_i P_{ij} \log P_{ij}$$
Entropy Rate of A Stationary Stochastic Process

Theorem

For a stationary stochastic process, $H(X)$ and $H'(X)$ exist and are equal:

For a stationary Markov chain, one has:

$$H(X) = H'(X) = \lim_{n \to \infty} H(X_n|X_{n-1}, \ldots, X_1) = \lim_{n \to \infty} H(X_n|X_{n-1})$$

$$= H(X_2|X_1)$$

where the initial state is drawn according to a stationary distribution $\mu$, i.e., $\mu_j = \sum_i \mu_i P_{ij} \forall j$. Can be more explicit?

Theorem

$$H(X) = H(X_2|X_1) = \sum_i \mu_i \left( \sum_j -P_{ij} \log P_{ij} \right) = - \sum_{ij} \mu_i P_{ij} \log P_{ij}$$
Entropy Rate of A Stationary Stochastic Process

Theorem

For a stationary stochastic process, $H(X)$ and $H'(X)$ exist and are equal:

For a stationary Markov chain, one has:

$$H(X) = H'(X) = \lim_{n \to \infty} H(X_n | X_{n-1}, \ldots, X_1) = \lim_{n \to \infty} H(X_n | X_{n-1})$$

$$= H(X_2 | X_1)$$

where the initial state is drawn according to a stationary distribution $\mu$, i.e., $\mu_j = \sum_i \mu_i P_{ij} \forall j$. Can be more explicit?

Theorem

$$H(X) = H(X_2 | X_1) = \sum_i \mu_i \left( \sum_j -P_{ij} \log P_{ij} \right) = -\sum_{ij} \mu_i P_{ij} \log P_{ij}$$
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
**Introduction**

- Now we will establish fundamental limit for the compression of information.

- **Basic idea of compression**: For a data sequence from a random source, data compression is achieved by assigning short descriptions to the most frequent outcomes, and longer descriptions to the less frequent outcomes.

- We will focus on finding the shortest average description length of a RVs.

- **Example with Morse code**:
  - Morse code is a method of transmitting textual information using only four symbols: a dot, a dash, a letter space, and a word space.
  - In Morse code, the most frequent symbol is represented by a single dot, i.e., letter $E$. 
Introduction

Now we will establish fundamental limit for the compression of information.

**Basic idea of compression:** *For a data sequence from a random source, data compression is achieved by assigning short descriptions to the most frequent outcomes, and longer descriptions to the less frequent outcomes.*

- We will focus on finding the shortest average description length of a RVs.

**Example with Morse code:**
- Morse code is a method of transmitting textual information using only four symbols: a dot, a dash, a letter space, and a word space.
- In Morse code, the most frequent symbol is represented by a single dot, i.e., letter *E*. 
Introduction

- Now we will establish fundamental limit for the compression of information.

- **Basic idea of compression**: *For a data sequence from a random source, data compression is achieved by assigning short descriptions to the most frequent outcomes, and longer descriptions to the less frequent outcomes.*

- We will focus on finding the shortest average description length of a RVs.

- Example with Morse code:
  - Morse code is a method of transmitting textual information using only four symbols: a dot, a dash, a letter space, and a word space.
  - In Morse code, the most frequent symbol is represented by a single dot, i.e., letter *E*. 
Introduction

- Now we will establish fundamental limit for the compression of information.

**Basic idea of compression:** *For a data sequence from a random source, data compression is achieved by assigning short descriptions to the most frequent outcomes, and longer descriptions to the less frequent outcomes.*

- We will focus on finding the shortest average description length of a RVs.

- Example with Morse code:
  - Morse code is a method of transmitting textual information using only four symbols: a dot, a dash, a letter space, and a word space.
  - In Morse code, the most frequent symbol is represented by a single dot, i.e., letter \( E \).
Introduction

Now we will establish fundamental limit for the compression of information.

**Basic idea of compression:** For a data sequence from a random source, data compression is achieved by assigning short descriptions to the most frequent outcomes, and longer descriptions to the less frequent outcomes.

We will focus on finding the shortest average description length of a RVs.

Example with Morse code:

- Morse code is a method of transmitting textual information using only four symbols: a dot, a dash, a letter space, and a word space.
- In Morse code, the most frequent symbol is represented by a single dot, i.e., letter $E$. 
What Covered?

- We first define notation of an instantaneous code and learn about Kraft inequality in codeword length.
- The first important result we will see: Expected description length is greater than equal to the entropy.
- We will examine different codes to show that
  - Entropy is the data compression limit.
  - Entropy is the number of bits needed in random number generation.
  - Any code achieving entropy turns out to be optimal from many points of view.
What Covered?

- We first define notation of an instantaneous code and learn about Kraft inequality in codeword length.
- The first important result we will see: *Expected description length is greater than equal to the entropy.*
- We will examine different codes to show that
  - Entropy is the data compression limit.
  - Entropy is the number of bits needed in random number generation.
  - Any code achieving entropy turns out to be optimal from many points of view.
What Covered?

- We first define notation of an instantaneous code and learn about Kraft inequality in codeword length.
- The first important result we will see: *Expected description length is greater than equal to the entropy.*
- We will examine different codes to show that
  - Entropy is the data compression limit.
  - Entropy is the number of bits needed in random number generation.
  - Any code achieving entropy turns out to be optimal from many points of view.
What Covered?

- We first define notation of an instantaneous code and learn about Kraft inequality in codeword length.
- The first important result we will see: Expected description length is greater than equal to the entropy.
- We will examine different codes to show that
  - Entropy is the data compression limit.
  - Entropy is the number of bits needed in random number generation.
  - Any code achieving entropy turns out to be optimal from many points of view.
What Covered?

- We first define notation of an instantaneous code and learn about Kraft inequality in codeword length.
- The first important result we will see: *Expected description length is greater than equal to the entropy.*
- We will examine different codes to show that
  - Entropy is the data compression limit.
  - Entropy is the number of bits needed in random number generation.
  - Any code achieving entropy turns out to be optimal from many points of view.
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Definition (Source Code)

A source code $C$ for a RV $X$ is a mapping from $X$, the range of $X$, to $D^*$, the set of finite-length strings of symbols from a $D$-ary alphabet. $C(x)$ used to denote the codeword corresponding to $x$ and $l(x)$ denotes the length of $C(x)$.

Example: $C(red) = 00$ and $C(blue) = 11$ is a source code for $X = \{red, blue\}$ with alphabet $D = \{0, 1\}$.

Definition (Expected Length)

The expected length $L(C)$ of a source code $C$ for a RV $X$ with PMF $p(x)$ is given by:

$$L(C) = \sum_{x \in X} p(x)l(x)$$
A source code $C$ for a RV $X$ is a mapping from $X$, the range of $X$, to $D^*$, the set of finite-length strings of symbols from a $D$-ary alphabet. $C(x)$ used to denote the codeword corresponding to $x$ and $l(x)$ denotes the length of $C(x)$.

Example: $C(red) = 00$ and $C(blue) = 11$ is a source code for $X = \{red, blue\}$ with alphabet $D = \{0, 1\}$.

The expected length $L(C)$ of a source code $C$ for a RV $X$ with PMF $p(x)$ is given by:

$$L(C) = \sum_{x \in X} p(x)l(x)$$
Source Code - Example 1

- We consider a RV $X$ with the following distribution and codeword assignment:

\[
\text{Pr}(X = 1) = \frac{1}{2}, \quad \text{codeword } C(1) = 0
\]

\[
\text{Pr}(X = 2) = \frac{1}{4}, \quad \text{codeword } C(2) = 10
\]

\[
\text{Pr}(X = 3) = \frac{1}{8}, \quad \text{codeword } C(3) = 110
\]

\[
\text{Pr}(X = 4) = \frac{1}{8}, \quad \text{codeword } C(4) = 111
\]

- The entropy $H(X) = 1.75$ bits and so is $L(C)$. For this code, any sequence of bits can be uniquely decoded into a sequence of symbols of $X$. For example, the string 0110111100110 is decoded as 1234213.
Source Code - Example 1

- We consider a RV $X$ with the following distribution and codeword assignment:
  
  $\Pr(X = 1) = \frac{1}{2}$, codeword $C(1) = 0$
  
  $\Pr(X = 2) = \frac{1}{4}$, codeword $C(2) = 10$
  
  $\Pr(X = 3) = \frac{1}{8}$, codeword $C(3) = 110$
  
  $\Pr(X = 4) = \frac{1}{8}$, codeword $C(4) = 111$

- The entropy $H(X) = 1.75$ bits and so is $L(C)$. For this code, any sequence of bits can be uniquely decoded into a sequence of symbols of $X$. For example, the string 0110111100110 is decoded as 1234213.
Source Code - Example 2

- We consider a RV $X$ with the following distribution and codeword assignment:

  \[
  \begin{align*}
  \Pr(X = 1) &= \frac{1}{3}, \quad \text{codeword } C(1) = 0 \\
  \Pr(X = 2) &= \frac{1}{3}, \quad \text{codeword } C(2) = 10 \\
  \Pr(X = 3) &= \frac{1}{3}, \quad \text{codeword } C(3) = 11
  \end{align*}
  \]

- The entropy $H(X) = 1.58\text{bits}$ and the average length $L(C) = 1.66\text{bits}$, which is larger than $H$. For this code, any sequence of bits can also be uniquely decoded into a sequence of symbols of $X$. 
Source Code - Example 2

We consider a RV $X$ with the following distribution and codeword assignment:

\[
\begin{align*}
\Pr(X = 1) &= \frac{1}{3}, \text{ codeword } C(1) = 0 \\
\Pr(X = 2) &= \frac{1}{3}, \text{ codeword } C(2) = 10 \\
\Pr(X = 3) &= \frac{1}{3}, \text{ codeword } C(3) = 11
\end{align*}
\]

The entropy $H(X) = 1.58$ bits and the average length $L(C) = 1.66$ bits, which is larger than $H$. For this code, any sequence of bits can also be uniquely decoded into a sequence of symbols of $X$.\]
Now we define more stringent conditions on codes.

Definition (Nonsingular code)

A code is said to be *nonsingular* if every element of the range of $X$ maps into a different string in $D^*$; that is

$$x \neq x' \Rightarrow C(c) \neq C(x')$$

- It suffices for an unambiguous description.
- But what if we wish to send a sequence of codeword? Using special symbol like comma is not really efficient.
- Then we now define the extension of a code.
Now we define more stringent conditions on codes.

**Definition (Nonsingular code)**

A code is said to be *nonsingular* if every element of the range of \(X\) maps into a different string in \(D^*\); that is

\[
x \neq x' \implies C(c) \neq C(x')
\]

- It suffices for an unambiguous description.
- But what if we wish to send a sequence of codeword? Using special symbol like comma is not really efficient.
- Then we now define the extension of a code.
Now we define more stringent conditions on codes.

**Definition (Nonsingular code)**

A code is said to be *nonsingular* if every element of the range of $X$ maps into a different string in $D^*$; that is

$$x \neq x' \Rightarrow C(c) \neq C(x')$$

- It suffices for an unambiguous description.
- But what if we wish to send a sequence of codeword? Using special symbol like comma is not really efficient.
- Then we now define the extension of a code.
Now we define more stringent conditions on codes.

**Definition (Nonsingular code)**

A code is said to be *nonsingular* if every element of the range of $X$ maps into a different string in $\mathcal{D}^*$; that is

\[ x \neq x' \implies C(c) \neq C(x') \]

- It suffices for an unambiguous description.
- But what if we wish to send a sequence of codeword? Using special symbol like comma is not really efficient.
- Then we now define the extension of a code.
Definitions

Definition (The extension $C^*$)

The extension $C^*$ of a code $C$ is the mapping from finite-length strings of $\mathcal{X}$ to finite-length strings of $\mathcal{D}$ defined by:

$$C(x_1x_2\ldots x_n) \triangleq C(x^n) = C(x_1)C(x_2)\ldots C(x_n)$$

where $C(x_1)C(x_2)\ldots C(x_n)$ indicates concatenation of the corresponding codewords.

Example: if $C(x_1) = 00$ and $C(x_2) = 11$ then $C(x_1x_2) = 0011$. 
Definitions

Definition (Uniquely decodable)

A code is called uniquely decodable if its extension is non-singular

- It means that any encoded string in a uniquely decodable code has only one possible source string producing it.
Example: Nonsingular

Is the following code nonsingular?

\[ x = 1, \ C(x) = 0 \]
\[ x = 2, \ C(x) = 010 \]
\[ x = 3, \ C(x) = 01 \]
\[ x = 4, \ C(x) = 10 \]

Is it uniquely decodable? What about \( C(21) \) and \( C(33) \)? Also, the string 010 has three possible source sequences, 2 or 14 or 31.
Example: Nonsingular

Is the following code nonsingular?

\[ x = 1, C(x) = 0 \]
\[ x = 2, C(x) = 010 \]
\[ x = 3, C(x) = 01 \]
\[ x = 4, C(x) = 10 \]

Is it uniquely decodable?  What about \( C(21) \) and \( C(33) \)? Also, the string 010 has three possible source sequences, 2 or 14 or 31.
Example: Nonsingular

Is the following code nonsingular?

\[
x = 1, C(x) = 0 \\
x = 2, C(x) = 010 \\
x = 3, C(x) = 01 \\
x = 4, C(x) = 10
\]

Is it uniquely decodable? What about \( C(21) \) and \( C(33) \)? Also, the string 010 has three possible source sequences, 2 or 14 or 31.
Example: Uniquely Decodable

Is the following uniquely decodable?

\[ x = 1, \ C(x) = 10 \]
\[ x = 2, \ C(x) = 00 \]
\[ x = 3, \ C(x) = 11 \]
\[ x = 4, \ C(x) = 110 \]

- It can be verified any extension code is nonsingular.
- However, still, one might need to look at the entire string to determine even the first symbol in the corresponding string.
Example: Uniquely Decodable

Is the following uniquely decodable?

\[ x = 1, C(x) = 10 \]
\[ x = 2, C(x) = 00 \]
\[ x = 3, C(x) = 11 \]
\[ x = 4, C(x) = 110 \]

- It can be verified any extension code is nonsingular.
- However, still, one might need to look at the entire string to determine even the first symbol in the corresponding string.
Example: Uniquely Decodable

Is the following uniquely decodable?

\[
\begin{align*}
x &= 1, \quad C(x) = 10 \\
x &= 2, \quad C(x) = 00 \\
x &= 3, \quad C(x) = 11 \\
x &= 4, \quad C(x) = 110
\end{align*}
\]

- It can be verified any extension code is nonsingular.
- However, still, one might need to look at the entire string to determine even the first symbol in the corresponding string.
Prefix or Instantaneous Code

Definition (Prefix Code)

A code is called *prefix code* or an *instantaneous code* if no codeword is a prefix of any other codeword.

- This code can be decoded without reference to future codewords: It is because the end of a codeword is immediately recognizable.
- It means any symbol $x_i$ can be decoded as soon as we come to the end of the corresponding codeword.
- This code is a *self-punctuating*: One can look down the sequence of code symbols and add the commas to separate the codewords without looking at later symbols.
Prefix or Instantaneous Code

Definition (Prefix Code)

A code is called *prefix code* or an *instantaneous code* if no codeword is a prefix of any other codeword.

- This code can be decoded without reference to future codewords: It is because the end of a codeword is immediately recognizable.
- It means any symbol $x_i$ can be decoded as soon as we come to the end of the corresponding codeword.
- This code is a *self-punctuating*: One can look down the sequence of code symbols and add the commas to separate the codewords without looking at later symbols.
Source Codes

Prefix or Instantaneous Code

Definition (Prefix Code)

A code is called *prefix code* or an *instantaneous code* if no codeword is a prefix of any other codeword.

- This code can be decoded without reference to future codewords: It is because the end of a codeword is immediately recognizable.
- It means any symbol $x_i$ can be decoded as soon as we come to the end of the corresponding codeword.
- This code is a *self-punctuating*: One can look down the sequence of code symbols and add the commas to separate the codewords without looking at later symbols.
Prefix or Instantaneous Code

Definition (Prefix Code)

A code is called *prefix code* or an *instantaneous code* if no codeword is a prefix of any other codeword.

- This code can be decoded without reference to future codewords: It is because the end of a codeword is immediately recognizable.
- It means any symbol $x_i$ can be decoded as soon as we come to the end of the corresponding codeword.
- This code is a *self-punctuating*: One can look down the sequence of code symbols and add the commas to separate the codewords without looking at later symbols.
With the prefix or instantaneous code, if we take the binary string 01011111010, it is parsed as 0, 10, 111, 110, and 10.
Example

<table>
<thead>
<tr>
<th>X</th>
<th>Singular</th>
<th>Nonsingular, But Not Uniquely Decodable</th>
<th>Uniquely Decodable, But Not Instantaneous</th>
<th>Instantaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>010</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>01</td>
<td>11</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>10</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

With the prefix or instantaneous code, if we take the binary string 0101111010, it is parsed as 0, 10, 111, 110, and 10.
Classes of Codes: The Nesting

- All codes
- Nonsingular codes
- Uniquely decodable codes
- Instantaneous codes
Prefix code can be represented as a code tree:
- For a $D$-ary code, each node has $D$ branches.
- Each codeword is represented by a leaf on the tree.

Example with binary code $C(A, B, C, D) = (0, 10, 110, 111)$:
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Certainly, we wish to construct a prefix code of minimum (expected) length.

However, we cannot always assign short codewords to all source symbols, given the prefix condition.

We have the following inequality that imposes the set of codeword lengths possible for prefix code:

Theorem (Kraft inequality)

For any instantaneous code (prefix code) over an alphabet size of $D$, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy:

$$\sum_{i} D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.
Certainly, we wish to construct a prefix code of minimum (expected) length.

However, we cannot always assign short codewords to all source symbols, given the prefix condition.

We have the following inequality that imposes the set of codeword lengths possible for prefix code:

**Theorem (Kraft inequality)**

For any instantaneous code (prefix code) over an alphabet size of $D$, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy:

$$\sum_i D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.
Certainly, we wish to construct a prefix code of minimum (expected) length. However, we cannot always assign short codewords to all source symbols, given the prefix condition. We have the following inequality that imposes the set of codeword lengths possible for prefix code:

**Theorem (Kraft inequality)**

For any instantaneous code (prefix code) over an alphabet size of $D$, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy:

$$\sum_{i} D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.
Certainly, we wish to construct a prefix code of minimum (expected) length. However, we cannot always assign short codewords to all source symbols, given the prefix condition. We have the following inequality that imposes the set of codeword lengths possible for prefix code:

**Theorem (Kraft inequality)**

For any instantaneous code (prefix code) over an alphabet size of \( D \), the codeword lengths \( l_1, l_2, \ldots, l_m \) must satisfy:

\[
\sum_{i} D^{-l_i} \leq 1
\]

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.
Converse Kraft Example

- Given a set of length \{2, 2, 3, 3, 3\}, can we construct a prefix code with binary alphabet using converse Kraft procedure?
- We can also apply the following equivalent method:
  - Put $l_i$ into ascending order and set
    
    $$c_i = \sum_i D^{-l_i} \quad \text{or} \quad c_i = \sum_i p(x_i)$$

- Then the codeword $i$th is the number $c_i$ rounded off to $l_i$ bits.
Converse Kraft Example

- Given a set of length \( \{2, 2, 3, 3, 3\} \), can we construct a prefix code with binary alphabet using converse Kraft procedure?
- We can also apply the following equivalent method:
  - Put \( l_i \) into ascending order and set
    \[
    c_i = \sum_i D^{-l_i} \quad \text{or} \quad c_i = \sum_i p(x_i)
    \]
  - Then the codeword \( i \)th is the number \( c_i \) rounded off to \( l_i \) bits.
Definition (Countably Infinite Set)

Any set which can be put in a one-to-one correspondence with the natural numbers (or integers) so that a prescription can be given for identifying its members one at a time is called a countably infinite (or denumerably infinite) set. Examples of countable sets include the integers, algebraic numbers, and rational numbers.
Theorem (Extended Kraft inequality)

For any countably infinite set of codewords that form a prefix code, the codeword lengths satisfy the extended Kraft inequality:

\[
\sum_{i=1}^{\infty} D^{-l_i} \leq 1
\]

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.

Note: the Kraft inequality holds also for infinite set.
Theorem (Extended Kraft inequality)

For any countably infinite set of codewords that form a prefix code, the codeword lengths satisfy the extended Kraft inequality:

$$\sum_{i=1}^{\infty} D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.

Note: the Kraft inequality holds also for infinite set.
Kraft Inequality Uniquely for Decodable Codes

In the sense of expected length, the set of uniquely decodable codes - while larger - does not improve upon instantaneous codes:

Theorem (McMillan)

The codeword lengths of any uniquely decodable $D$-ary code must satisfy the Kraft inequality:

$$\sum_{i} D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.

Note: the Kraft inequality holds also for infinite set.
Kraft Inequality Uniquely for Decodable Codes

In the sense of expected length, the set of uniquely decodable codes - while larger - does not improve upon instantaneous codes:

Theorem (McMillan)

The codeword lengths of any uniquely decodable $D$-ary code must satisfy the Kraft inequality:

$$\sum_i D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.

Note: the Kraft inequality holds also for infinite set.
Kraft Inequality Uniquely for Decodable Codes

In the sense of expected length, the set of uniquely decodable codes - while larger - does not improve upon instantaneous codes:

**Theorem (McMillan)**

The codeword lengths of any uniquely decodable $D$-ary code must satisfy the Kraft inequality:

$$\sum_{i} D^{-l_i} \leq 1$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.

Note: the Kraft inequality holds also for infinite set.
Outline

1. Review of Entropy Rate
2. Data Compression: Introduction
3. Source Codes
4. Kraft Inequality
5. Optimal Codes & Bounds
7. Huffman Codes
Optimal Codes - Problem Statement

- We know earlier that any prefix code must satisfy the Kraft inequality.
- Now, we are interested in the problem of finding the prefix code with the minimum expected length. This code, if exists, is called *optimal code*.
- It is therefore equivalent to finding the set of lengths $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality and whose expected length $L = \sum_i p_i l_i$ is less than the expected length of any other prefix code.
We know earlier that any prefix code must satisfy the Kraft inequality.

Now, we are interested in the problem of finding the prefix code with the minimum expected length. This code, if exists, is called *optimal code*.

It is therefore equivalent to finding the set of lengths \( l_1, l_2, \ldots, l_m \) satisfying the Kraft inequality and whose expected length \( L = \sum_i p_i l_i \) is less than the expected length of any other prefix code.
We know earlier that any prefix code must satisfy the Kraft inequality.

Now, we are interested in the problem of finding the prefix code with the minimum expected length. This code, if exists, is called **optimal code**.

It is therefore equivalent to finding the set of lengths $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality and whose expected length $L = \sum_i p_i l_i$ is less than the expected length of any other prefix code.
Optimal Codes

Optimal code: Prefix code that minimizes the expected codeword length. It is a solution to minimizing

$$L = \sum_{i=1}^{m} p_i l_i$$

over all integers $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality:

$$\sum_{i=1}^{m} D^{-l_i} \leq 1$$
We relax the integer constraints and assume equality in the constraint. Then use the Lagrangian:

$$J = \sum_i p_i l_i + \lambda \left( \sum_i D^{-l_i} \right)$$

Differentiating with respect to $l_i$, we have:

$$\frac{\partial J}{\partial l_i} = p_i - \lambda D^{-l_i} \log_e D = 0$$

It means that $D^{-l_i} = \frac{p_i}{\lambda \log_e D}$. Using the constraint, we have

$$\lambda = \frac{1}{\log_e D} \quad \text{and} \quad p_i = D^{-l_i}$$
Optimal Codes - Solution

- We relax the integer constraints and assume equality in the constraint. Then use the Lagrangian:

\[
J = \sum_i p_i l_i + \lambda \left( \sum_i D^{-l_i} \right)
\]

- Differentiating with respect to \( l_i \), we have:

\[
\frac{\partial J}{\partial l_i} = p_i - \lambda D^{-l_i} \log_e D = 0
\]

- It means that \( D^{-l_i} = \frac{p_i}{\lambda \log_e D} \). Using the constraint, we have

\[
\lambda = \frac{1}{\log_e D}
\]

and

\[
p_i = D^{-l_i}
\]
So we have the optimal code length:

\[ l_i^* = - \log_D p_i \]

This non-integer choice of codeword lengths yields expected codeword length:

\[ L^* = \sum_{i} l_i^* = - \sum_{i} p_i \log_D p_i = H_D(X) \]

Of course, we will not always achieve the codeword lengths above. We should choose a set of integer lengths that are close to the optimal set.

Is the above solution global minimum?
So we have the optimal code length:

\[ l_i^* = - \log_D p_i \]

This non-integer choice of codeword lengths yields expected codeword length:

\[ L^* = \sum_{i} l_i^* = - \sum_{i} p_i \log_D p_i = H_D(X) \]

Of course, we will not always achieve the codeword lengths above. We should choose a set of integer lengths that are close to the optimal set.

Is the above solution global minimum?
Optimal Codes - Solution

- So we have the optimal code length:
  \[ l_i^* = - \log_D p_i \]

- This non-integer choice of codeword lengths yields expected codeword length:
  \[ L^* = \sum_{p_i} l_i^* = - \sum_i p_i \log_D p_i = H_D(X) \]

- Of course, we will not always achieve the codeword lengths above. We should choose a set of integer lengths that are close to the optimal set.

- Is the above solution global minimum?
Theorem (Lower bound on the expected length)

The expected length $L$ of any prefix $D$-ary code for a rv $X$ is greater than or equal the entropy $H_D(X)$; that is:

$$L \geq H_D(X)$$

with equality if and only if $D^{-l_i} = p_i$. 

$$L^* = \sum_{i} l_i^* = - \sum_{i} p_i \log_D p_i = H_D(X)$$
Proof of Lower Bound
**D** – *adic* Distribution

**Definition**

A probability distribution is called *D* – *adic* if each of the probabilities is equal to $D^{-n}$ for some $n$.

It is obvious that we have equality in the lower bound if and only if the distribution is *D* – *adic*. We can then have the following procedure to find optimal code:

- Given a distribution of $X$, we find *D* – *adic* distribution closest in the relative entropy to it.
- Construct the code as in the converse Kraft sample earlier.

However, searching closest *D* – *adic* is not obvious. Later, we will study good suboptimal procedure (Shannon-Fano coding) and another simple procedure (Huffman coding) for finding optimal one.
**Definition**

A probability distribution is called $D – adic$ if each of the probabilities is equal to $D^{-n}$ for some $n$.

It is obvious that we have equality in the lower bound if and only if the distribution is $D – adic$. We can then have the following procedure to find optimal code:

1. Given a distribution of $X$, we find $D – adic$ distribution closest in the relative entropy to it.
2. Construct the code as in the converse Kraft sample earlier.

However, searching closest $D – adic$ is not obvious. Later, we will study good suboptimal procedure (Shannon-Fano coding) and another simple procedure (Huffman coding) for finding optimal one.
A probability distribution is called $D –adic$ if each of the probabilities is equal to $D^{−n}$ for some $n$.

It is obvious that we have equality in the lower bound if and only if the distribution is $D –adic$. We can then have the following procedure to find optimal code:

- Given a distribution of $X$, we find $D –adic$ distribution closest in the relative entropy to it.
- Construct the code as in the converse Kraft sample earlier.

However, searching closest $D –adic$ is not obvious. Later, we will study good suboptimal procedure (Shannon-Fano coding) and another simple procedure (Huffman coding) for finding optimal one.
Shannon Code and Upper Bound

Earlier, we learned that the optimal choice of codeword lengths $l_i = \log_D \frac{1}{p_i}$ yields $L = H_D$. However, $\log_D \frac{1}{p_i}$ might not be an integer and we don't have a valid code.

Now, what happens if we consider the following integer word-length assignments:

$$l_i = \lceil \log_D \frac{1}{p_i} \rceil$$

where $\lceil x \rceil$ is the smallest integer $\geq x$.

It is easy to verify this set of lengths satisfies Kraft inequality, since

$$\sum_i D^{-\lceil \log_D \frac{1}{p_i} \rceil} \leq \sum_i D^{-\log_D \frac{1}{p_i}} = \sum p_i = 1$$

It means that we can have a valid prefix code: Shannon Code!!!
Optimal Codes & Bounds

Shannon Code and Upper Bound

- Earlier, we learned that the optimal choice of codeword lengths \( l_i = \log_D \frac{1}{p_i} \) yields \( L = H_D \). However, \( \log_D \frac{1}{p_i} \) might not be an integer and we dont have a valid code.

- Now, what happen if we consider the following integer word-length assignments:

\[
l_i = \lceil \log_D \frac{1}{p_i} \rceil
\]

where \( \lceil x \rceil \) is the smallest integer \( \geq x \).

- It is easy to very find this set of lengths satisfies Kraft inequality, since

\[
\sum_i D^{-\lceil \log_D \frac{1}{p_i} \rceil} \leq \sum_i D^{-\log_D \frac{1}{p_i}} = \sum_i p_i = 1
\]

- It means that we can have a valid prefix code: Shannon Code!!!!
Shannon Code and Upper Bound

- Earlier, we learned that the optimal choice of codeword lengths $l_i = \log_D \frac{1}{p_i}$ yields $L = H_D$. However, $\log_D \frac{1}{p_i}$ might not be an integer and we don't have a valid code.

- Now, what happen if we consider the following integer word-length assignments:

  $$l_i = \lceil \log_D \frac{1}{p_i} \rceil$$

  where $\lceil x \rceil$ is the smallest integer $\geq x$.

- It is easy to very find this set of lengths satisfies Kraft inequality, since

  $$\sum_i D^{-\lceil \log_D \frac{1}{p_i} \rceil} \leq \sum_i D^{-\log_D \frac{1}{p_i}} = \sum_{p_i} = 1$$

- It means that we can have a valid prefix code: Shannon Code!!!!
Shannon Code: Construction Method

- For a given distribution, round up optimal code length
  \[ l_i = \lceil \log_D \frac{1}{p_i} \rceil. \]
- This choice of code lengths satisfies Kraft inequality, hence prefix code exists.
- Put \( l_i \) into ascending order and set
  \[ c_i = \sum_i D^{-l_i} \quad \text{or} \quad c_i = \sum_i p(x_i) \]
- Then the codeword \( i \)th is the number \( c_i \) rounded off to \( l_i \) bits.
Shannon Code: Construction Method

- For a given distribution, round up optimal code length
  \[ l_i = \lceil \log_D \frac{1}{p_i} \rceil. \]
- This choice of code lengths satisfies Kraft inequality, hence prefix code exists.
- Put \( l_i \) into ascending order and set
  \[ c_i = \sum D^{-l_i} \quad \text{or} \quad c_i = \sum p(x_i) \]
- Then the codeword \( i \)th is the number \( c_i \) rounded off to \( l_i \) bits.
Shannon Code: Example

Example 1:

\[ p(X) = [0.5, 0.25, 0.125, 0.125] \]

\[ - \log_2 p(X) = [1, 2, 3, 3] \]

\[ l_i = - \left\lfloor \log_2 \frac{1}{p_i} \right\rfloor = [1, 2, 3, 3] \]

\[ L_C = H(X) = 1.75 \text{ bits} \]

What is the code?
Example 1:

\[ p(X) = [0.5, 0.25, 0.125, 0.125] \]

\[ - \log_2 p(X) = [1, 2, 3, 3] \]

\[ l_i = -\left\lceil \log_2 \frac{1}{p_i} \right\rceil = [1, 2, 3, 3] \]

\[ L_C = H(X) = 1.75 \text{bits} \]

What is the code?
Upper Bound

**We know that we can always construct a valid code, which is Shannon Code.**

- Now, let see *how good the code is* by examining the expected length for Shannon Code with \( l_i = \lceil \log_D \frac{1}{p_i} \rceil \).

  - It is easy to verified that

  \[
  \log_D \frac{1}{p_i} \leq l_i < \log_D \frac{1}{p_i} + 1
  \]

  - Then by multiplying by \( p_i \) and summing over \( i \), we obtain:

  \[
  H_D(X) \leq L < H_D(X) + 1
  \]

  Recall that \( H_D(X) = - \sum_i p_i \log_D p_i \).
Upper Bound

- We know that we can always construct a valid code, which is *Shannon Code*.

- Now, let see *how good the code is* by examining the expected length for Shannon Code with $l_i = \lceil \log_D \frac{1}{p_i} \rceil$.
  - It is easy to verified that
    \[
    \log_D \frac{1}{p_i} \leq l_i < \log_D \frac{1}{p_i} + 1
    \]
  - Then by multiplying by $p_i$ and summing over $i$, we obtain:
    \[
    H_D(X) \leq L < H_D(X) + 1
    \]

Recall that $H_D(X) = -\sum_i p_i \log_D p_i$. 
Upper Bound

- We know that we can always construct a valid code, which is Shannon Code.
- Now, let see how good the code is by examining the expected length for Shannon Code with $l_i = \lceil \log_D \frac{1}{p_i} \rceil$.
  - It is easy to verify that
    \[
    \log_D \frac{1}{p_i} \leq l_i < \log_D \frac{1}{p_i} + 1
    \]
  - Then by multiplying by $p_i$ and summing over $i$, we obtain:
    \[
    H_D(X) \leq L < H_D(X) + 1
    \]
- Recall that $H_D(X) = - \sum_i p_i \log_D p_i$. 
Since *Shannon Code* might not be optimal, we can have the following theorem regarding the optimal code:

**Theorem**

Let \( l_1^*, \ldots, l_m^* \) be optimal codeword lengths for a source distribution \( p \) and a \( D \)-ary alphabet, and let \( L^* \) be the associated expected length of an optimal code (\( L^* = \sum_i p_i l_i^* \)). Then

\[
H_D(X) \leq L^* < H_D(X) + 1
\]

- We see that there is an overhead that is almost 1 bit, due to the fact that \( \log \frac{1}{p_i} \) is not always an integer.
- We can reduce the overhead by using *block coding*: spreading over many symbols.
Upper and Lower Bounds on Optimal Code

Since *Shannon Code* might not be optimal, we can have the following theorem regarding the optimal code:

**Theorem**

Let \( l_1^*, \ldots, l_m^* \) be optimal codeword lengths for a source distribution \( p \) and a \( D \)-ary alphabet, and let \( L^* \) be the associated expected length of an optimal code (\( L^* = \sum_i p_i l_i^* \)). Then

\[
H_D(X) \leq L^* < H_D(X) + 1
\]

- We see that there is an overhead that is almost 1 bit, due to the fact that \( \log \frac{1}{p_i} \) is not always an integer.
- We can reduce the overhead by using *block coding*: spreading over many symbols.
Since *Shannon Code* might not be optimal, we can have the following theorem regarding the optimal code:

**Theorem**

Let $l_1^*, \ldots, l_m^*$ be optimal codeword lengths for a source distribution $p$ and a $D$-ary alphabet, and let $L^*$ be the associated expected length of an optimal code ($L^* = \sum_i p_i l_i^*$). Then

$$H_D(X) \leq L^* < H_D(X) + 1$$

We see that there is an overhead that is almost 1 bit, due to the fact that $\log \frac{1}{p_i}$ is not always an integer.

We can reduce the overhead by using *block coding*: spreading over many symbols.
Since *Shannon Code* might not be optimal, we can have the following theorem regarding the optimal code:

**Theorem**

Let $l_1^*, \ldots, l_m^*$ be optimal codeword lengths for a source distribution $p$ and a $D$-ary alphabet, and let $L^*$ be the associated expected length of an optimal code ($L^* = \sum_i p_i l_i^*$). Then

$$H_D(X) \leq L^* < H_D(X) + 1$$

- We see that there is an overhead that is almost 1 bit, due to the fact that $\log \frac{1}{p_i}$ is not always an integer.
- We can reduce the overhead by using *block coding*: spreading over many symbols.
Block Coding with I.I.D. Sources

- Now assume we send $n$ symbols drawn i.i.d according to $p(X)$ in a block, so that we have a “supersymbol” from $\mathcal{X}^n$. Also, let $D = 2$ hereafter.

- Define $L_n$ be the expected length per input symbol:

$$L_n = \frac{1}{n} \sum_{p(x_1, \ldots, x_n)l(x_1, \ldots, x_n)} = \frac{1}{n} E \{ l(X_1, \ldots, X_n) \}$$

where $l(x_1, \ldots, x_n)$ is the length associated with $(x_1, \ldots, x_n)$.

- Apply the bounds to the code, we have:

$$H(X_1, \ldots, X_n) \leq E \{ l(X_1, \ldots, X_n) \} < H(X_1, \ldots, X_n) + 1$$

- Since RVs are i.i.d, we then have:

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

We can achieve the length close to entropy.
Block Coding with I.I.D. Sources

- Now assume we send $n$ symbols drawn i.i.d according to $p(X)$ in a block, so that we have a “supersymbol” from $\mathcal{X}^n$. Also, let $D = 2$ hereafter.
- Define $L_n$ be the expected length per input symbol:

$$L_n = \frac{1}{n} \sum_{p(x_1, \ldots, x_n)l(x_1, \ldots, x_n)} = \frac{1}{n} E \{l(X_1, \ldots, X_n)\}$$

where $l(x_1, \ldots, x_n)$ is the length associated with $(x_1, \ldots, x_n)$.
- Apply the bounds to the code, we have:

$$H(X_1, \ldots, X_n) \leq E \{l(X_1, \ldots, X_n)\} < H(X_1, \ldots, X_n) + 1$$

- Since RVs are i.i.d, we then have:

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

We can achieve the length close to entropy.
Block Coding with I.I.D. Sources

Now assume we send \( n \) symbols drawn i.i.d according to \( p(X) \) in a block, so that we have a “supersymbol” from \( \mathcal{X}^n \). Also, let \( D = 2 \) hereafter.

Define \( L_n \) be the expected length per input symbol:

\[
L_n = \frac{1}{n} \sum_{p(x_1,\ldots,x_n)l(x_1,\ldots,x_n)} = \frac{1}{n}E\{l(X_1,\ldots,X_n)\}
\]

where \( l(x_1,\ldots,x_n) \) is the length associated with \( (x_1,\ldots,x_n) \).

Apply the bounds to the code, we have:

\[
H(X_1,\ldots,X_n) \leq E\{l(X_1,\ldots,X_n)\} < H(X_1,\ldots,X_n) + 1
\]

Since RVs are i.i.d, we then have:

\[
H(X) \leq L_n < H(X) + \frac{1}{n}
\]

We can achieve the length close to entropy.
We can apply the same argument to a sequence of symbols from a stochastic process and obtain:

\[
\frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}
\]

We know that \( \frac{H(X_1, \ldots, X_n)}{n} \rightarrow H(X) \): Expected length tends to the entropy rate.
Stationary Stochastic Process

We can apply the same argument to a sequence of symbol from stochastic process and obtain:

\[
\frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}
\]

We know that \( H(X_1, \ldots, X_n)/n \to H(X) \): Expected length tends to the entropy rate.
Theorem

The minimum expected codeword length per symbol satisfies

\[
\frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}.
\]

Moreover, if we have a stationary process,

\[ L_n^* \to H(X) \]

where \( H(X) \) is the entropy rate of the process.

This theorem provides another justification for the definition of entropy rate: It is the expected number of bits per symbol required to describe the process.
Wrong Distribution

Now, we are interested in the case of the code design with wrong distribution. This might happen when we try to estimate a distribution for the unknown one.

We consider Shannon code with assignment \( l_i = \lceil \log_D \frac{1}{q_i} \rceil \) with distribution \( q(X) \). Suppose the true distribution is \( p(X) \). Certainly, we cannot achieve the expected length \( L \approx H(p) = -\sum_i p_i \log p_i \).

The question is: What is the increase in expected length due to incorrect distribution?
Wrong Distribution

Now, we are interested in the case of the code design with wrong distribution. This might happen when we try to estimate a distribution for the unknown one.

We consider Shannon code with assignment \( l_i = \lceil \log_D \frac{1}{q_i} \rceil \) with distribution \( q(X) \). Suppose the true distribution is \( p(X) \). Certainly, we cannot achieve the expected length \( L \approx H(p) = - \sum_i p_i \log p_i \).

The question is: What is the increase in expected length due to incorrect distribution?
Wrong Distribution

Now, we are interested in the case of the code design with wrong distribution. This might happen when we try to estimate a distribution for the unknown one.

We consider Shannon code with assignment \( l_i = \lceil \log_D \frac{1}{q_i} \rceil \) with distribution \( q(X) \). Suppose the true distribution is \( p(X) \). Certainly, we cannot achieve the expected length \( L \approx H(p) = -\sum_i p_i \log p_i \).

The question is: What is the increase in expected length due to incorrect distribution?
Wrong Distribution

Theorem

The expected length under \( p(X) \) of the code assignment \( l_i = \lceil \log \frac{1}{q_i} \rceil \) with \( q(X) \) satisfies:

\[
H(p) + D(p\|q) \leq E_p \{ l(X) \} < H(p) + D(p\|q) + 1
\]

Proof.

- We see that the penalty by using wrong distribution is the relative entropy \( D(p\|q) \).
- \( D(p\|q) \) has a concrete interpretation as the increase in descriptive complexity due to incorrect information.
Wrong Distribution

Theorem

The expected length under $p(X)$ of the code assignment $l_i = \lceil \log \frac{1}{q_i} \rceil$ with $q(X)$ satisfies:

$$H(p) + D(p||q) \leq E_p \{ l(X) \} < H(p) + D(p||q) + 1$$

Proof.

- We see that the penalty by using wrong distribution is the relative entropy $D(p||q)$.
- $D(p||q)$ has a concrete interpretation as the increase in descriptive complexity due to incorrect information.
Wrong Distribution

Theorem

The expected length under $p(X)$ of the code assignment $l_i = \lceil \log \frac{1}{q_i} \rceil$ with $q(X)$ satisfies:

$$H(p) + D(p\|q) \leq E_p \{ l(X) \} < H(p) + D(p\|q) + 1$$

Proof.

- We see that the penalty by using wrong distribution is the relative entropy $D(p\|q)$.
- $D(p\|q)$ has a concrete interpretation as the increase in descriptive complexity due to incorrect information.
We know about Shannon code and study the bounds. It appears that Shannon code is good most of the time.

We have the following theorem:

**Theorem**

Let $l(x)$ be the codeword lengths associated with the Shannon code, and let $l'(x)$ be the codeword lengths associated with any other uniquely decodable code. Then:

$$
\Pr \left( l(X) \geq l'(X) + c \right) \leq \frac{1}{2^{c-1}}
$$

**Proof.**

No other code can do much better than Shannon code most of the time.
Shannon Code Competitive Optimality

- We know about Shannon code and study the bounds. It appears that Shannon code is good most of the time.
- We have the following theorem:

**Theorem**

Let $l(x)$ be the codeword lengths associated with the Shannon code, and let $l'(x)$ be the codeword lengths associated with any other uniquely decodable code. Then:

$$\Pr\left( l(X) \geq l'(X) + c \right) \leq \frac{1}{2^{c-1}}$$

**Proof.**

No other code can do much better than Shannon code most of the time.
We know about Shannon code and study the bounds. It appears that Shannon code is good most of the time.

We have the following theorem:

**Theorem**

Let $l(x)$ be the codeword lengths associated with the Shannon code, and let $l'(x)$ be the codeword lengths associated with any other uniquely decodable code. Then:

$$
Pr \left( l(X) \geq l'(X) + c \right) \leq \frac{1}{2^{c-1}}
$$

**Proof.**

No other code can do much better than Shannon code most of the time.
Dyadic Competitive Optimality

- We can strengthen the result by focusing on the dyadic distribution. Recall that $p(X)$ is dyadic if $\log \frac{1}{p(x)}$ is an integer for all $x$.

**Theorem**

For a Dyadic probability mass function $p(X)$, let $l(x)$ be the word lengths of the binary Shannon code for the source, and let $l'(x)$ be the lengths of any other uniquely decodable binary code for the source. Then,

$$\Pr\left(l(X) < l'(X)\right) \geq \Pr\left(l(X) > l'(X)\right)$$

with equality if and only if $l'(X) = l(X)$ for all $x$. Thus, the code length assignment is uniquely competitively optimal.
Earlier, we showed that we can obtain Shannon code using
\[ l(x) = \lceil \log_D \frac{1}{p(x)} \rceil. \]

Now we are going to describe a simple constructive code
using cumulative distribution function (CMF) to allot
codewords, referred to as Shannon-Fano-Elias code.

Without loss of generality, assume \( \mathcal{X} = \{1, 2, \ldots, m\} \). We also assume \( p(x) > 0 \).
Earlier, we showed that we can obtain Shannon code using 
\[ l(x) = \lceil \log_D \frac{1}{p(x)} \rceil. \]

Now we are going to describe a simple constructive code using cumulative distribution function (CMF) to allot codewords, referred to as Shannon-Fano-Elias code.

Without loss of generality, assume \( \mathcal{X} = \{1, 2, \ldots, m\} \). We also assume \( p(x) > 0 \).
Earlier, we showed that we can obtain Shannon code using
\[ l(x) = \lceil \log_D \frac{1}{p(x)} \rceil. \]
Now we are going to describe a simple constructive code using cumulative distribution function (CMF) to allot codewords, referred to as *Shannon-Fano-Elias code*.
Without loss of generality, assume \( \mathcal{X} = \{1, 2, \ldots, m\} \). We also assume \( p(x) > 0 \).
CDF and Modified CDF

- Recall: $F_X(x) \sum_{a\leq x} p(a)$, which consists of steps size of $p(x)$.
- Let define the modified CDF:
  \[
  \bar{F}_X(x) = \sum_{a<x} p(a) + \frac{1}{2} p(x)
  \]
  which is sum of probabilities of all symbols smaller than $x$, plus half of probability of symbol $x$: midpoint of the step.
CDF and Modified CDF

- Recall: $F_X(x) \sum_{a \leq x} p(a)$, which consists of steps size of $p(x)$.
- Let define the modified CDF:
  $$\bar{F}_X(x) = \sum_{a < x} p(a) + \frac{1}{2} p(x)$$
  which is sum of probabilities of all symbols smaller than $x$, plus half of probability of symbol $x$: midpoint of the step.
Shannon-Fano-Elias Code

- \( \bar{F}_X(x) \), or for convenience, \( \bar{F}(x) \), is positive and \( \bar{F}(a) \neq \bar{F}(b) \) if \( a \neq b \). Therefore, we can determine \( x \) if we know \( \bar{F}(x) \).

- Generally, \( \bar{F}(x) \) is real, not efficient as a code. So here is **Shannon-Fano-Elias Code**

  Let \( l(x) = \lceil \log_D \frac{1}{p(x)} \rceil + 1 \). The codeword of \( x \) is obtained by rounding off \( \bar{F}(x) \) to \( l(x) \) bits, denoted as \( \lfloor \bar{F}(x) \rfloor_{l(x)} \).

- We now show it is a valid code.
Shannon-Fano-Elias Code

\[ \bar{F}_X(x), \text{ or for convenience, } \bar{F}(x), \text{ is positive and } \bar{F}(a) \neq \bar{F}(b) \text{ if } a \neq b. \] Therefore, we can determine \( x \) if we know \( \bar{F}(x) \).

Generally, \( \bar{F}(x) \) is real, not efficient as a code. So here is Shannon-Fano-Elias Code

- Let \( l(x) = \left\lceil \log_D \frac{1}{p(x)} \right\rceil + 1 \). The codeword of \( x \) is obtained by rounding off \( \bar{F}(x) \) to \( l(x) \) bits, denoted as \( \lfloor \bar{F}(x) \rfloor l(x) \).
- We now show it is a valid code.
Shannon-Fano-Elias Code

- By definition, \( \bar{F}(x) - \lfloor \bar{F}(x) \rfloor l(x) < \frac{1}{2^{l(x)}} \).
- Furthermore, since \( l(x) = \lceil \log_D \frac{1}{p(x)} \rceil + 1 \),

\[
\frac{1}{2^{l(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x - 1)
\]

It means \( \lfloor \bar{F}(x) \rfloor l(x) \) lies within the step corresponding to \( x \). Equivalently, the codeword is sufficient to identify the symbol.

- Are intervals \( \lfloor \bar{F}(x) \rfloor l(x), \lfloor \bar{F}(x) \rfloor l(x) + \frac{1}{2^{l(x)}} \) disjoint?
  - Lower end is in the lower half of the step
  - Upper end is below the top of the step
  - It means the interval lies entirely within the step (corresponding symbol).

Therefore, code is prefix.
By definition, \( \bar{F}(x) - \lfloor \bar{F}(x) \rfloor l(x) < \frac{1}{2^{l(x)}} \).

Furthermore, since \( l(x) = \lceil \log_D \frac{1}{p(x)} \rceil + 1 \),

\[
\frac{1}{2^{l(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x - 1)
\]

It means \( \lfloor \bar{F}(x) \rfloor l(x) \) lies within the step corresponding to \( x \). Equivalently, the codeword is sufficient to identify the symbol.

Are intervals \( \lfloor \bar{F}(x) \rfloor l(x), \lfloor \bar{F}(x) \rfloor l(x) + \frac{1}{2^{l(x)}} \) disjoint?

- Lower end is in the lower half of the step
- Upper end is below the top of the step
- It means the interval lies entirely within the step (corresponding symbol).

Therefore, code is prefix.
Observe that the construction procedure does not require symbols to be ordered in terms of probability.

Finally, we can have a bound on the expected length:

\[
L = \sum_x p(x) l(x) = \sum_x p(x) \left( \left\lceil \log_D \frac{1}{p(x)} \right\rceil + 1 \right) < H(X) + 2
\]

This coding scheme achieves an average codeword length that is within 2 bits of entropy.
Now we are going to construct a code from the distribution \( p = [0.25, 0.5, 0.125, 0.125] \). Observe that the distribution is dyadic.

How about distribution is not dyadic \( p = [0.25, 0.5, 0.2, 0.15, 0.15] \)?
Now we are going to construct a code from the distribution $p = [0.25, 0.5, 0.125, 0.125]$. Observe that the distribution is dyadic.

How about distribution is not dyadic $p = [0.25, 0.5, 0.2, 0.15, 0.15]$?
Now we are going to learn Huffman codes. These codes are optimal (shortest expected length) prefix code. They can be constructed by a simple algorithm discovered by Huffman.

D. A. Huffman, “A method for the construction of minimum redundancy codes"  

Before going in further details, we first introduce some characteristics of optimal codes.
Now we are going to learn Huffman codes. These codes are optimal (shortest expected length) prefix code. They can be constructed by a simple algorithm discovered by Huffman.

D. A. Huffman, “A method for the construction of minimum redundancy codes”  

Before going in further details, we first introduce some characteristics of optimal codes.
Simple Property of Optimal Codes

- We first note that if $C$ is an optimal code and $p_j > p_k$, then $l_j \leq l_k$: Frequent symbols are coded with short codewords. But why?
- We consider $C'$ that is equal to $C$, except that codewords $C_j$ and $C_k$ are swapped.
- Now we are going to calculate the expected length difference:

$$L(C') - L(C) = p_k l_j + p_j l_k - p_k l_k - p_j l_j = (p_j - p_k)(l_k - l_j)$$

- Obviously, if $p_j > p_k$, $l_j \leq l_k$. Otherwise, $C$ is not optimal.
Simple Property of Optimal Codes

- We first note that if \( C \) is an optimal code and \( p_j > p_k \), then \( l_j \leq l_k \): Frequent symbols are coded with short codewords. But why?
- We consider \( C' \) that is equal to \( C \), except that codewords \( C_j \) and \( C_k \) are swapped.
- Now we are going to calculate the expected length difference:

\[
L(C') - L(C) = p_k l_j + p_j l_k - p_k l_k - p_j l_j = (p_j - p_k)(l_k - l_j)
\]
- Obviously, if \( p_j > p_k \), \( l_j \leq l_k \). Otherwise, \( C \) is not optimal.
Simple Property of Optimal Codes

- We first note that if $C$ is an optimal code and $p_j > p_k$, then $l_j \leq l_k$: Frequent symbols are coded with short codewords. But why?

- We consider $C'$ that is equal to $C$, except that codewords $C_j$ and $C_k$ are swapped.

- Now we are going to calculate the expected length difference:

$$L(C') - L(C) = p_k l_j + p_j l_k - p_k l_k - p_j l_j = (p_j - p_k)(l_k - l_j)$$

- Obviously, if $p_j > p_k$, $l_j \leq l_k$. Otherwise, $C$ is not optimal.
Characterization of Optimal Prefix Codes

We have the following results for binary code. Extension to $D$-ary ones is straightforward.

**Lemma**

For any distribution, there exists an optimal prefix (with minimum expected length) that satisfies the following properties:

1. The lengths are ordered inversely with the probabilities, i.e., if $p_j > p_k$, then $l_j \leq l_k$.
2. The two longest codewords have the same length.
3. Two of the longest codewords differ only in the last bit and correspond to the two least likely symbols.

**Proof.**
Huffman Code

**Definition**

Let $C$ be an instantaneous code which satisfies the above lemma and assume, without less of generality, that $p_1 \geq p_2 \geq \ldots \geq p_m$. This code is a Huffman code.

We will show the optimality of these codes later!!
Construction of Huffman Code

A Huffman code can be obtained by repeatedly “merging” the last two symbols, assigning to them the “last codeword minus the last bit”, and reordering the symbols in order to have non-increasing probabilities.

In particular, to find an Huffman code we repeat the following procedure until we end up with only two symbols:

1. Replace $x_{m-1}$ and $x_m$ by a new symbol $t_{m-1}$ having probability $p_{m-1} + p_m$.
2. Assign to $t_{m-1}$ the codeword obtained by removing the last bit in $x_{m-1}$ or $x_m$.
3. Reorder $x_1, x_2, \ldots, x_{m-2}, t_{m-1}$ according to non increasing probabilities.

We finally have the trivial solution: Assigning 0 to one symbol and 1 to the other. At this point, we can trace back for the original code.
Construction of Huffman Code

A Huffman code can be obtained by repeatedly “merging” the last two symbols, assigning to them the “last codeword minus the last bit”, and reordering the symbols in order to have non-increasing probabilities.

In particular, to find an Huffman code we repeat the following procedure until we end up with only two symbols:

1. Replace $x_{m-1}$ and $x_m$ by a new symbol $t_{m-1}$ having probability $p_{m-1} + p_m$.
2. Assign to $t_{m-1}$ the codeword obtained by removing the last bit in $x_{m-1}$ or $x_m$.
3. Reorder $x_1, x_2, \ldots, x_{m-2}, t_{m-1}$ according to non increasing probabilities.

We finally have the trivial solution: Assigning 0 to one symbol and 1 to the other. At this point, we can trace back for the original code.
Construction of Huffman Code

- A Huffman code can be obtained by repeatedly “merging” the last two symbols, assigning to them the “last codeword minus the last bit”, and reordering the symbols in order to have non-increasing probabilities.

- In particular, to find an Huffman code we repeat the following procedure until we end up with only two symbols:
  1. Replace \( x_{m-1} \) and \( x_m \) by a new symbol \( t_{m-1} \) having probability \( p_{m-1} + p_m \).
  2. Assign to \( t_{m-1} \) the codeword obtained by removing the last bit in \( x_{m-1} \) or \( x_m \).
  3. Reorder \( x_1, x_2, \ldots, x_{m-2}, t_{m-1} \) according to non increasing probabilities.

We finally have the trivial solution: Assigning 0 to one symbol and 1 to the other. At this point, we can trace back for the original code.
Construction of Huffman Code

A Huffman code can be obtained by repeatedly “merging” the last two symbols, assigning to them the “last codeword minus the last bit”, and reordering the symbols in order to have non-increasing probabilities.

In particular, to find an Huffman code we repeat the following procedure until we end up with only two symbols:

1. Replace $x_{m-1}$ and $x_m$ by a new symbol $t_{m-1}$ having probability $p_{m-1} + p_m$.
2. Assign to $t_{m-1}$ the codeword obtained by removing the last bit in $x_{m-1}$ or $x_m$.
3. Reorder $x_1, x_2, \ldots, x_{m-2}, t_{m-1}$ according to non-increasing probabilities.

We finally have the trivial solution: Assigning 0 to one symbol and 1 to the other. At this point, we can trace back for the original code.
A Huffman code can be obtained by repeatedly “merging” the last two symbols, assigning to them the “last codeword minus the last bit”, and reordering the symbols in order to have non-increasing probabilities.

In particular, to find an Huffman code we repeat the following procedure until we end up with only two symbols:

1. Replace $x_{m-1}$ and $x_m$ by a new symbol $t_{m-1}$ having probability $p_{m-1} + p_m$.
2. Assign to $t_{m-1}$ the codeword obtained by removing the last bit in $x_{m-1}$ or $x_m$.
3. Reorder $x_1, x_2, \ldots, x_{m-2}, t_{m-1}$ according to non increasing probabilities.

We finally have the trivial solution: Assigning 0 to one symbol and 1 to the other. At this point, we can trace back for the original code.
Consider a RV $X$ taking values in the set $\{1, 2, 3, 4, 5\}$ with probabilities $0.25, 0.25, 0.2, 0.15, 0.15$, respectively. We expect the optimal binary code for $X$ to have the longest codewords assigned to the symbols 4 and 5.

<table>
<thead>
<tr>
<th>Codeword Length</th>
<th>Codeword</th>
<th>$X$</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>01</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.55</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.45</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>000</td>
<td>4</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>001</td>
<td>5</td>
<td>0.15</td>
</tr>
</tbody>
</table>

The expected length is $L = 2.3$ and $H(X) = 2.286$. 
Huffman Code: English Alphabet

<table>
<thead>
<tr>
<th>a_i</th>
<th>p_i</th>
<th>\log_2 \frac{1}{p_i}</th>
<th>l_i</th>
<th>c(a_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.0575</td>
<td>4.1</td>
<td>4</td>
<td>0000</td>
</tr>
<tr>
<td>b</td>
<td>0.0128</td>
<td>6.3</td>
<td>6</td>
<td>001000</td>
</tr>
<tr>
<td>c</td>
<td>0.0263</td>
<td>5.2</td>
<td>5</td>
<td>00101</td>
</tr>
<tr>
<td>d</td>
<td>0.0285</td>
<td>5.1</td>
<td>5</td>
<td>10000</td>
</tr>
<tr>
<td>e</td>
<td>0.0913</td>
<td>3.5</td>
<td>4</td>
<td>1100</td>
</tr>
<tr>
<td>f</td>
<td>0.0173</td>
<td>5.9</td>
<td>6</td>
<td>111000</td>
</tr>
<tr>
<td>g</td>
<td>0.0133</td>
<td>6.2</td>
<td>6</td>
<td>001001</td>
</tr>
<tr>
<td>h</td>
<td>0.0313</td>
<td>5.0</td>
<td>5</td>
<td>10001</td>
</tr>
<tr>
<td>i</td>
<td>0.0599</td>
<td>4.1</td>
<td>4</td>
<td>1001</td>
</tr>
<tr>
<td>j</td>
<td>0.0006</td>
<td>10.7</td>
<td>10</td>
<td>1101000000</td>
</tr>
<tr>
<td>k</td>
<td>0.0084</td>
<td>6.9</td>
<td>7</td>
<td>1010000</td>
</tr>
<tr>
<td>l</td>
<td>0.0335</td>
<td>4.9</td>
<td>5</td>
<td>11101</td>
</tr>
<tr>
<td>m</td>
<td>0.0235</td>
<td>5.4</td>
<td>6</td>
<td>110101</td>
</tr>
<tr>
<td>n</td>
<td>0.0596</td>
<td>4.1</td>
<td>4</td>
<td>0001</td>
</tr>
<tr>
<td>o</td>
<td>0.0689</td>
<td>3.9</td>
<td>4</td>
<td>1011</td>
</tr>
<tr>
<td>p</td>
<td>0.0192</td>
<td>5.7</td>
<td>6</td>
<td>111001</td>
</tr>
<tr>
<td>q</td>
<td>0.0008</td>
<td>10.3</td>
<td>9</td>
<td>110100001</td>
</tr>
<tr>
<td>r</td>
<td>0.0508</td>
<td>4.3</td>
<td>5</td>
<td>11011</td>
</tr>
<tr>
<td>s</td>
<td>0.0567</td>
<td>4.1</td>
<td>4</td>
<td>0011</td>
</tr>
<tr>
<td>t</td>
<td>0.0706</td>
<td>3.8</td>
<td>4</td>
<td>1111</td>
</tr>
<tr>
<td>u</td>
<td>0.0334</td>
<td>4.9</td>
<td>5</td>
<td>10101</td>
</tr>
<tr>
<td>v</td>
<td>0.0069</td>
<td>7.2</td>
<td>8</td>
<td>11010001</td>
</tr>
<tr>
<td>w</td>
<td>0.0119</td>
<td>6.4</td>
<td>7</td>
<td>1101001</td>
</tr>
<tr>
<td>x</td>
<td>0.0073</td>
<td>7.1</td>
<td>7</td>
<td>1010001</td>
</tr>
<tr>
<td>y</td>
<td>0.0164</td>
<td>5.9</td>
<td>6</td>
<td>101001</td>
</tr>
<tr>
<td>z</td>
<td>0.0007</td>
<td>10.4</td>
<td>10</td>
<td>1101000001</td>
</tr>
<tr>
<td>−</td>
<td>0.1928</td>
<td>2.4</td>
<td>2</td>
<td>01</td>
</tr>
</tbody>
</table>
Huffman Code: Ternary Example

Still consider a RV $X$ taking values in the set $\{1, 2, 3, 4, 5\}$ with probabilities $0.25, 0.25, 0.2, 0.15, 0.15$. Now we construct *ternary code*, i.e., $D = 3$ by combining the three least likely symbols into one supersymbol:

<table>
<thead>
<tr>
<th>Codeword $X$</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1</td>
<td>0.25</td>
</tr>
<tr>
<td>2 2</td>
<td>0.25</td>
</tr>
<tr>
<td>00 3</td>
<td>0.2</td>
</tr>
<tr>
<td>01 4</td>
<td>0.15</td>
</tr>
<tr>
<td>02 5</td>
<td>0.15</td>
</tr>
</tbody>
</table>

The expected length is of 1.5 ternary digits.
It satisfies three important properties of optimal code:

1. The lengths are ordered inversely with the probabilities, i.e., if \( p_j > p_k \), then \( l_j \leq l_k \).

2. The two longest codewords have the same length.

3. Two of the longest codewords differ only in the last bit and correspond to the two least likely symbols.

Is it optimal?
Huffman Code: Optimal Prefix Code?

It satisfies three important properties of optimal code:

1. The lengths are ordered inversely with the probabilities, i.e., if $p_j > p_k$, then $l_j \leq l_k$.
2. The two longest codewords have the same length.
3. Two of the longest codewords differ only in the last bit and correspond to the two least likely symbols.

Is it optimal?
Assume there exists \( m > 2 \) such that Huffman code \( C_m \) is the first sub-optimal code.

An optimal \( C'_m \) must have \( L'_m < L_m \).

Rearrange the symbols with longest codes in \( C'_m \) so two lowest \( p_m \) and \( p_{m-1} \) differ only in the last digit.

Merge \( x_m \) and \( x_{m-1} \) to create a new code \( C'_{m-1} \) as in Huffman procedure.

It can be easy to verified that \( L'_{m-1} = L'_m - p_m - p_{m-1} \). Also, \( L_{m-1} = L_m - p_m - p_{m-1} \). It means \( C_{m-1} \) is also suboptimal: Contradiction.
Assume there exists \( m > 2 \) such that Huffman code \( C_m \) is the first sub-optimal code.

An optimal \( C'_m \) must have \( L'_m < L_m \).

- Rearrange the symbols with longest codes in \( C'_m \) so two lowest \( p_m \) and \( p_{m-1} \) differ only in the last digit.

- Merge \( x_m \) and \( x_{m-1} \) to create a new code \( C'_{m-1} \) as in Huffman procedure.

- It can be easy to verified that \( L'_{m-1} = L'_m - p_m - p_{m-1} \). Also, \( L'_{m-1} = L_m - p_m - p_{m-1} \). It means \( C_{m-1} \) is also suboptimal: Contradiction.
Optimality of Huffman Code

Assume there exists $m > 2$ such that Huffman code $C_m$ is the first sub-optimal code.

An optimal $C'_m$ must have $L'_m < L_m$.

Rearrange the symbols with longest codes in $C'_m$ so two lowest $p_m$ and $p_{m-1}$ differ only in the last digit.

Merge $x_m$ and $x_{m-1}$ to create a new code $C'_{m-1}$ as in Huffman procedure.

It can be easily verified that $L'_{m-1} = L'_m - p_m - p_{m-1}$. Also, $L_{m-1} = L_m - p_m - p_{m-1}$. It means $C_{m-1}$ is also suboptimal: Contradiction.
Assume there exists $m > 2$ such that Huffman code $C_m$ is the first sub-optimal code.

An optimal $C'_m$ must have $L'_m < L_m$.

Rearrange the symbols with longest codes in $C'_m$ so two lowest $p_m$ and $p_{m-1}$ differ only in the last digit.

Merge $x_m$ and $x_{m-1}$ to create a new code $C'_{m-1}$ as in Huffman procedure.

It can be easy to verified that $L'_{m-1} = L'_m - p_m - p_{m-1}$. Also, $L_{m-1} = L_m - p_m - p_{m-1}$. It means $C_{m-1}$ is also suboptimal: Contradiction.
Optimality of Huffman Code

- Assume there exists $m > 2$ such that Huffman code $C_m$ is the first sub-optimal code.
- An optimal $C'_m$ must have $L'_m < L_m$.
- Rearrange the symbols with longest codes in $C'_m$ so two lowest $p_m$ and $p_{m-1}$ differ only in the last digit.
- Merge $x_m$ and $x_{m-1}$ to create a new code $C'_{m-1}$ as in Huffman procedure.
- It can be easy to verified that $L'_{m-1} = L'_m - p_m - p_{m-1}$. Also, $L_{m-1} = L_m - p_m - p_{m-1}$. It means $C_{m-1}$ is also suboptimal: Contradiction.
Theorem (Optimality of Huffman coding)

Huffman coding is optimal; that is, if $C^*$ is a Huffman code and $C'$ is any other uniquely decodable code, then $L(C^*) \leq L(C')$. 
Huffman Code: Disadvantages

- Huffman is an optimal symbol code for an ensemble, but this is not the end of the story.
- For unchanging ensemble, i.e., changing probabilities, a Huffman code may be convenient. However, it often changes:
  - We are compressing text, then the symbol frequencies will vary with context: in English the letter $u$ is much more probable after a $q$ than after an $e$. Our knowledge of these context-dependent symbol frequencies will also change as we learn the statistical properties of the text source.
  - One brute-force approach would be to recompute the Huffman code: Not feasible.
  - In overall, Huffman code has many defects for practical purposes.
Huffman Code: Disadvantages

- Huffman is an optimal symbol code for an ensemble, but this is not the end of the story.
- For unchanging ensemble, i.e., changing probabilities, a Huffman code may be convenient. However, it often changes:
  - We are compressing text, then the symbol frequencies will vary with context: in English the letter \( u \) is much more probable after a \( q \) than after an \( e \). Our knowledge of these context-dependent symbol frequencies will also change as we learn the statistical properties of the text source.
  - One brute-force approach would be to recompute the Huffman code: Not feasible.
  - In overall, Huffman code has many defects for practical purposes.
Huffman is an optimal symbol code for an ensemble, but this is not the end of the story. For unchanging ensemble, i.e., changing probabilities, a Huffman code may be convenient. However, it often changes:

- We are compressing text, then the symbol frequencies will vary with context: in English the letter $u$ is much more probable after a $q$ than after an $e$. Our knowledge of these context-dependent symbol frequencies will also change as we learn the statistical properties of the text source.

- One brute-force approach would be to recompute the Huffman code: Not feasible.

- In overall, Huffman code has many defects for practical purposes.
Huffman Code: Disadvantages

- Huffman is an optimal symbol code for an ensemble, but this is not the end of the story.
- For unchanging ensemble, i.e., changing probabilities, a Huffman code may be convenient. However, it often changes:
  - We are compressing text, then the symbol frequencies will vary with context: in English the letter $u$ is much more probable after a $q$ than after an $e$. Our knowledge of these context-dependent symbol frequencies will also change as we learn the statistical properties of the text source.
  - One brute-force approach would be to recompute the Huffman code: Not feasible.
  - In overall, Huffman code has many defects for practical purposes.
The idea of the Shannon-Fano-Elias coding can be extended to blocks of source symbols: *Arithmetic coding*.

*Arithmetic coding* is a beautiful method that goes hand in hand with the philosophy that compression of data from a source entails probabilistic modelling of that source.

*Arithmetic coding*: the best compression methods for text files use arithmetic coding, and several state-of-the-art image compression systems use it too.
Beyond Huffman Code: Arithmetic Coding

The idea of the Shannon-Fano-Elias coding can be extended to blocks of source symbols: *Arithmetic coding*.

*Arithmetic coding* is a beautiful method that goes hand in hand with the philosophy that compression of data from a source entails probabilistic modelling of that source.

*Arithmetic coding*: the best compression methods for text files use arithmetic coding, and several state-of-the-art image compression systems use it too.
The idea of the Shannon-Fano-Elias coding can be extended to blocks of source symbols: *Arithmetic coding*.

*Arithmetic coding* is a beautiful method that goes hand in hand with the philosophy that compression of data from a source entails probabilistic modelling of that source.

*Arithmetic coding*: the best compression methods for text files use arithmetic coding, and several state-of-the-art image compression systems use it too.
Beyond Huffman Code: Lempel-Ziv Coding

- **Lempel-Ziv Coding** is a “universal” method, designed under the philosophy that we would like a single compression algorithm that will do a reasonable job for any source.

- **Lempel-Ziv Coding** is widely used and often effective.

- To my knowledge, ZIP and gzip use the DEFLATE algorithm for compression: Combination of Lempel-Ziv coding and Huffman coding.
Beyond Huffman Code: Lempel-Ziv Coding

- *Lempel-Ziv Coding* is a “universal” method, designed under the philosophy that we would like a single compression algorithm that will do a reasonable job for any source.

- *Lempel-Ziv Coding* is widely used and often effective.

- To my knowledge, ZIP and gzip use the DEFLATE algorithm for compression: Combination of Lempel-Ziv coding and Huffman coding.
Lempel-Ziv Coding is a “universal” method, designed under the philosophy that we would like a single compression algorithm that will do a reasonable job for any source.

Lempel-Ziv Coding is widely used and often effective.

To my knowledge, ZIP and gzip use the DEFLATE algorithm for compression: Combination of Lempel-Ziv coding and Huffman coding.
Thank you!