Class Road Map - IT-I

- L1 (9/25): Overview, Communications, Information, Entropy
- L2 (9/30): Entropy, Mutual Information, KL-Divergence
- L3 (10/2): More KL, Jensen, more Venn, Log Sum, Data Proc. Inequality
- L4 (10/7): Data Proc. Ineq., thermodynamics, Stats, Fano,
- L5 (10/9): M. of Conv, AEP,
- L6 (10/14): AEP, Source Coding, Types
- L7 (10/16): Types, Univ. Src Coding, Stoc. Procs, Entropy Rates
- L8 (10/21): Entropy rates, HMMs, Coding
- L9 (10/23): Kraft ineq., Shannon Codes, Kraft ineq. II, Huffman

- L10 (10/28): Discussion of Fall. 1
- L11 (10/30):
- L12 (11/4):
- LXX (11/6): In class midterm exam
- LXX (11/11): Veterans Day holiday
- L13 (11/13):
- L14 (11/18):
- L15 (11/20):
- L16 (11/25):
- L17 (11/27):
- L18 (12/2):
- L19 (12/4):
- LXX (12/10): Final exam

Finals Week: December 9th–13th.
Cumulative Outstanding Reading

- Read chapters 1 and 2 in our book (Cover & Thomas, “Information Theory”) (including Fano’s inequality).
- Read chapters 3 and 4 in our book (Cover & Thomas, “Information Theory”).
- Read sections 11.1 through 11.3 in our book (Cover & Thomas, “Information Theory”).
- Read chapter 4 in our book (Cover & Thomas, “Information Theory”).
Homework

- Homework 1 on our assignment dropbox (https://canvas.uw.edu/courses/1319497/assignments), was due Tuesday, Oct 8th, 11:55pm.
- Homework 2 on our assignment dropbox (https://canvas.uw.edu/courses/1319497/assignments), due Friday 10/18/2019, 11:45pm.
- Homework 3 on our assignment dropbox (https://canvas.uw.edu/courses/1319497/assignments), due Tuesday 10/29/2019, 11:45pm.

Office hours 9:30 - midnight tonight on zoom.
What is entropy of this random walk

So, the entropy of the random walk is

\[ H(X) = (\text{overall edge uncertainty}) - (\text{overall node uncertainty in stationary condition}) \]

Intuition: As node entropy decreases while keeping edge uncertainty constant, fewer nodes are hubs. Hubs that remain are widely connected (since edge entropy is fixed).

In such case (few well connected hubs), it is likely one will land on such a hub (in a random walk) and then will be faced with a wide variety of choice as to where to go next \( \Rightarrow \) increase in overall uncertainty \( H(X) = H(X_t|X_{t-1}) \) of the walk.

If node entropy goes up with edge entropy fixed, many nodes are hubs all with relatively low connectivity, hitting them doesn’t provide much choice \( \Rightarrow \) random walk entropy \( H(X) = H(X_t|X_{t-1}) \) goes down.
We want to develop practical coding algorithms that still approach, or achieve, the entropy limit.

They might use the distribution $p(x)$ which is either given or is estimated in some way.

We won’t get into any details on how to estimate $p(x)$ (that is a density estimation problem) but we assume we either have it or some approximation.

We will ultimately look, however, at what happens if the true distribution is $p(x)$ and we use $q(x)$ instead.
Source Code

**Definition 9.2.2 (source code)**

A source code $C$ for r.v. $X$ is a mapping

$$C : \mathcal{X} \rightarrow \mathcal{D}^* = \{\mathcal{D} \cup (\mathcal{D} \times \mathcal{D}) \cup (\mathcal{D} \times \mathcal{D} \times \mathcal{D}) \cup \ldots\} \quad (9.24)$$

from $\mathcal{X}$ to $\mathcal{D}^*$, the set of finite strings from a $D$-ary alphabet. $C(x)$ is the codeword corresponding to $x$, and $\ell(x)$ is the length of the codeword.

**Example 9.2.3**

Let $\mathcal{X} = \{\text{red}, \text{blue}\}$. Then a code might be $C(\text{red}) = 00$ and $C(\text{blue}) = 11$, which would be a binary code for $\mathcal{D} = \{0, 1\}$.

**Definition 9.2.4 (expected length)**

The expected length $L(C)$ of code $C$ for r.v. $X$ with distribution $p(x)$ is

$$L(C) = \sum_x p(x)\ell(x) \quad (9.25)$$
Definition 9.2.3 (non-singular)

A code is said to be non-singular if every element of the range of $X$ (i.e., all elements of $X$) maps to a different string in $\mathcal{D}^*$. I.e.,

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j) \quad (9.24)$$

- We can view this as a mapping. It is less strict than onto but sufficient for being able to decode individual symbols.

- Note that $C_1$ above is singular.
Our goal, and definition of Code Extension

Before going further, note: our goal is to send or store a sequence of code words for a sequence of symbols.

A non-singular code could be unique if \( \exists \) a comma between code words (e.g., Morse code is such that there is a space).

In general, however, it is better to have a self punctuating or instantaneous code.

**Definition 9.2.3 (code extension)**

A code extension \( C^* \) of \( C \) is a mapping from finite length strings of \( D \), defined as:

\[
C^*(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n)
\]  \hfill (9.24)

Note that there are no commas in the extension, rather concatenation.

Ex: If \( C(x_1) = 0 \) and \( C(x_2) = 1 \) then \( C(x_1, x_2) = 01 \).
Code types: uniquely decodable

**Definition 9.2.3 (uniquely decodable)**

A code $C$ with extension $C^*$ is **uniquely decodable** if the extension $C^*$ is non-singular.

- $C_1$ singular. Extension of $C_{||}$ singular so $C_{||}$ not uniquely decodable.
- But how long must we wait until we know the source? In some even uniquely decodable cases, we might need to wait until the end.
- Ex: consider the code

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C(x)$</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>110</td>
</tr>
</tbody>
</table>

Is this code uniquely decodable?
Yes.
Prefix codes

Definition 9.2.3 (prefix code)

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

- We know the end of a codeword because it can’t be a prefix of any other codeword.
- Code in previous page is not prefix free, 11 was a prefix of 110 so we couldn’t decide between 11 or 110 until we could count the number of zeros.
- A prefix code is self-punctuating (since there are implicit punctuation marks between codewords).
- Prefix code $\Rightarrow$ uniquely decodable. But (as we saw) uniquely decodable $\nRightarrow$ prefix code.
Goal is to find a code with the shortest possible expected length.

From the above code class, we might think that we want to use codes from the largest class possible (since we might think we’re more likely to get shorter codes).

We can do better than entropy with non-singular codes, but we want lossless encoding $x = \text{ungzip}(\text{gzip}(x))$. 
Theorem 9.3.1 (Kraft inequality)

For any instantaneous code (prefix code) over alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy

$$
\sum_{i} D^{-\ell_i} \leq 1 \quad \text{(9.1)}
$$
Theorem 9.3.1 (Kraft inequality)

For any instantaneous code (prefix code) over alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy

$$\sum_{i} D^{-\ell_i} \leq 1 \quad (9.1)$$

Conversely, given a set of codeword lengths satisfying the above inequality, $\exists$ an instantaneous code with these word lengths.
**Theorem 9.3.1 (Kraft inequality)**

*For any instantaneous code (prefix code) over alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy*

$$
\sum_{i} D^{-\ell_i} \leq 1 \quad (9.1)
$$

*Conversely, given a set of codeword lengths satisfying the above inequality, $\exists$ an instantaneous code with these word lengths.*

- Note: converse says there exists a code with these lengths, not that all codes with these lengths will satisfy the inequality.
Kraft inequality

Theorem 9.3.1 (Kraft inequality)

For any instantaneous code (prefix code) over alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy

$$\sum_{i} D^{-\ell_i} \leq 1 \quad (9.1)$$

Conversely, given a set of codeword lengths satisfying the above inequality, $\exists$ an instantaneous code with these word lengths.

- Note: converse says there exists a code with these lengths, not that all codes with these lengths will satisfy the inequality.
- Key point: for $\ell_i$ satisfying Kraft, no further restriction imposed by also wanting a prefix code, so we might as well use a prefix code (assuming it is easy to find given the lengths)
Theorem 9.3.1 (Kraft inequality)

For any instantaneous code (prefix code) over alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy

$$\sum_i D^{-\ell_i} \leq 1 \quad (9.1)$$

Conversely, given a set of codeword lengths satisfying the above inequality, $\exists$ an instantaneous code with these word lengths.

- Note: converse says there exists a code with these lengths, not that all codes with these lengths will satisfy the inequality.
- Key point: for $\ell_i$ satisfying Kraft, no further restriction imposed by also wanting a prefix code, so we might as well use a prefix code (assuming it is easy to find given the lengths)
- Connects code existence to mathematical property on lengths!
  
Given Kraft lengths, can construct an instantaneous code (as we will see). Given lengths, can compute $E[\ell]$ and compare with $H$. 
proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:
proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:
proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:

![Diagram of a $D$-ary tree with codewords and prefixes]
proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:

```
  1
 /|
/  |

  2
 /|
/  |

  D
```

Codewords correspond to leaves.
Path from root to leaf determines a codeword.
Prefix condition: won't get to a codeword until we get to a leaf (no descendants of codewords are codewords).
proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:

  ![Diagram of a D-ary tree]

  - Codewords correspond to leaves
proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:
  - Codewords correspond to leaves
  - Path from root to leaf determines a codeword
Kraft inequality

proof of Kraft inequality.

- Represent set of codes on a $D$-ary (not necessarily balanced) tree:

- Codewords correspond to leaves
- Path from root to leaf determines a codeword
- Prefix condition: won't get to a codeword until we get to a leaf (no descendants of codewords are codewords)

...
\[ \ell_{\text{max}} = \max_i (\ell_i) \] is the length of the longest codeword.
... proof of Kraft inequality cont.

- $\ell_{\text{max}} = \max_i (\ell_i)$ is the length of the longest codeword.
- We can expand the full-tree down to depth $\ell_{\text{max}}$. 

...
... proof of Kraft inequality cont.

- $\ell_{\text{max}} = \max_i (\ell_i)$ is the length of the longest codeword.
- We can expand the full-tree down to depth $\ell_{\text{max}}$. 
Kraft inequality

... proof of Kraft inequality cont.

- $\ell_{\text{max}} = \max_i(\ell_i)$ is the length of the longest codeword.
- We can expand the full-tree down to depth $\ell_{\text{max}}$.

![Diagram showing a full-tree expanded down to depth $\ell_{\text{max}}$.]
\[ \ell_{\text{max}} = \max_i (\ell_i) \] is the length of the longest codeword.

We can expand the full-tree down to depth \( \ell_{\text{max}} \).

Some nodes at that level \( \ell_{\text{max}} \) are either:

- \( \ell_1 \) codewords,
- \( \ell_2 \) descendants of codewords, or
- neither.
Kraft inequality

\[ \ell_{\text{max}} = \max_i (\ell_i) \] is the length of the longest codeword.

We can expand the full-tree down to depth \( \ell_{\text{max}} \).

Some nodes at that level \( \ell_{\text{max}} \) are either:

1. codewords,
... proof of Kraft inequality cont.

- \( \ell_{\text{max}} = \max_i (\ell_i) \) is the length of the longest codeword.
- We can expand the full-tree down to depth \( \ell_{\text{max}} \).
  
  Some nodes at that level \( \ell_{\text{max}} \) are either:

  1. codewords,
  2. descendants of codewords, or
... proof of Kraft inequality cont.

- \( \ell_{\text{max}} = \max_i (\ell_i) \) is the length of the longest codeword.
- We can expand the full-tree down to depth \( \ell_{\text{max}} \).

Some nodes at that level \( \ell_{\text{max}} \) are either:

1. codewords,
2. descendants of codewords, or
3. neither
... proof of Kraft inequality cont.

- $\ell_{\text{max}} = \max_i(\ell_i)$ is the length of the longest codeword.
- We can expand the full-tree down to depth $\ell_{\text{max}}$. Some nodes at that level $\ell_{\text{max}}$ are either:
  1. codewords,
  2. descendants of codewords, or
  3. neither
- Consider a codeword $i$ at level $\ell_i$ in tree (so it has length $\ell_i$).
Kraft inequality

... proof of Kraft inequality cont.

- $\ell_{\text{max}} = \max_i(\ell_i)$ is the length of the longest codeword.
- We can expand the full-tree down to depth $\ell_{\text{max}}$.

Some nodes at that level $\ell_{\text{max}}$ are either:

1. codewords,
2. descendants of codewords, or
3. neither

- Consider a codeword $i$ at level $\ell_i$ in tree (so it has length $\ell_i$).
- Then, there are $D^{\ell_{\text{max}} - \ell_i}$ descendants in the tree at level $\ell_{\text{max}}$. 
Kraft inequality

... proof of Kraft inequality cont.

- \( \ell_{\text{max}} = \max_i (\ell_i) \) is the length of the longest codeword.
- We can expand the full-tree down to depth \( \ell_{\text{max}} \).
  
  Some nodes at that level \( \ell_{\text{max}} \) are either:
  
  1. codewords,
  2. descendants of codewords, or
  3. neither

- Consider a codeword \( i \) at level \( \ell_i \) in tree (so it has length \( \ell_i \)).
- Then, there are \( D^{\ell_{\text{max}}-\ell_i} \) descendants in the tree at level \( \ell_{\text{max}} \).
- Because of prefix condition, descendants of code \( i \) at level \( \ell_i \) are disjoint from descendants of code \( j \) at level \( \ell_j \) when \( i \neq j \) (i.e., descendant sets for different codewords are disjoint).

...
Kraft inequality

... proof of Kraft inequality cont.

- \( \ell_{\text{max}} = \max_i(\ell_i) \) is the length of the longest codeword.
- We can expand the full-tree down to depth \( \ell_{\text{max}} \).
  - Some nodes at that level \( \ell_{\text{max}} \) are either:
    1. codewords,
    2. descendants of codewords, or
    3. neither
- Consider a codeword \( i \) at level \( \ell_i \) in tree (so it has length \( \ell_i \)).
- Then, there are \( D^{\ell_{\text{max}}-\ell_i} \) descendants in the tree at level \( \ell_{\text{max}} \).
- Because of prefix condition, descendants of code \( i \) at level \( \ell_i \) are disjoint from descendants of code \( j \) at level \( \ell_j \) when \( i \neq j \) (i.e., descendant sets for different codewords are disjoint).
- Also, total number of nodes in set of all descendants is \( \leq D^{\ell_{\text{max}}} \). ...
... proof of Kraft inequality cont.

- All of the above implies:

\[
\sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \implies \sum_i D^{-\ell_i} \leq 1 \quad (9.2)
\]
...proof of Kraft inequality cont.

- All of the above implies:

\[
\sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \quad \Rightarrow \quad \sum_i D^{-\ell_i} \leq 1 \quad (9.2)
\]

- Conversely: given codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ satisfying Kraft inequality (we must construct a prefix code with these lengths).
All of the above implies:

\[
\sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \quad \Rightarrow \quad \sum_i D^{-\ell_i} \leq 1 \quad (9.2)
\]

Conversely: given codeword lengths \(\ell_1, \ell_2, \ldots, \ell_m\) satisfying Kraft inequality (we must construct a prefix code with these lengths).

Consider a full \(D\)-ary tree of depth \(\ell_{\text{max}}\) with \(D^{\ell_{\text{max}}}\) terminal nodes.
Kraft inequality

... proof of Kraft inequality cont.

- All of the above implies:

\[
\sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \quad \Rightarrow \quad \sum_i D^{-\ell_i} \leq 1 \tag{9.2}
\]

- Conversely: given codeword lengths \(\ell_1, \ell_2, \ldots, \ell_m\) satisfying Kraft inequality (we must construct a prefix code with these lengths).

- Consider a full \(D\)-ary tree of depth \(\ell_{\text{max}}\) with \(D^{\ell_{\text{max}}}\) terminal nodes.

  - @ level 0, \(\exists\) fraction 1 of the descendants at each node at that level;
  - @ level 1, \(\exists\) fraction \(1/D\) descendants at each node at that level;
  - @ level 2, \(\exists\) fraction \(1/D^2\) \ldots
Kraft inequality

... proof of Kraft inequality cont.

- All of the above implies:

$$\sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \implies \sum_i D^{-\ell_i} \leq 1 \quad (9.2)$$

- Conversely: given codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ satisfying Kraft inequality (we must construct a prefix code with these lengths).

- Consider a full $D$-ary tree of depth $\ell_{\text{max}}$ with $D^{\ell_{\text{max}}}$ terminal nodes.

  @ level 0, $\exists$ fraction 1 of the descendants at each node at that level;
  @ level 1, $\exists$ fraction $1/D$ descendants at each node at that level;
  @ level 2, $\exists$ fraction $1/D^2$ ...

- In general, at each level $i \in [0, \ell_{\text{max}}]$ in tree, there is a fraction $D^{-i}$ terminal nodes that are descendants that stem from each of the $D^{i}$ nodes at level $i$. 
...proof of Kraft inequality cont.

- Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) ascending to \((s_1, s_2, \ldots, s_m)\) with 
  \(s_1 \leq s_2 \leq \cdots \leq s_m\). Note there are as many lengths as there are 
  codewords.
...proof of Kraft inequality cont.

- Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) ascending to \((s_1, s_2, \ldots, s_m)\) with \(s_1 \leq s_2 \leq \cdots \leq s_m\). Note there are as many lengths as there are codewords.

- For length \(s_1\) chose any node at level \(s_1\) to indicate the codeword.

...
Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) ascending to \((s_1, s_2, \ldots, s_m)\) with \(s_1 \leq s_2 \leq \cdots \leq s_m\). Note there are as many lengths as there are codewords.

For length \(s_1\) chose any node at level \(s_1\) to indicate the codeword.

To ensure prefix free property, the node becomes a terminal node, thus eliminating a fraction \(D^{-s_1}\) of the terminal nodes at depth \(\ell_{\text{max}}\) (which would have been potential code words of longer length, but now they are out of the running).
Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) ascending to \((s_1, s_2, \ldots, s_m)\) with \(s_1 \leq s_2 \leq \cdots \leq s_m\). Note there are as many lengths as there are codewords.

For length \(s_1\) chose any node at level \(s_1\) to indicate the codeword.

To ensure prefix free property, the node becomes a terminal node, thus eliminating a fraction \(D_{s_1}\) of the terminal nodes at depth \(\ell_{\text{max}}\) (which would have been potential code words of longer length, but now they are out of the running).

Next: chose any remaining node at level \(s_2\) (we have \((D_{s_1} - 1)D_{s_2-s_1} > 0\) choices at this point) for next codeword, thus eliminating a fraction \(D_{s_2}\) of the nodes.
Kraft inequality

...proof of Kraft inequality cont.

- Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) ascending to \((s_1, s_2, \ldots, s_m)\) with \(s_1 \leq s_2 \leq \cdots \leq s_m\). Note there are as many lengths as there are codewords.

- For length \(s_1\) chose any node at level \(s_1\) to indicate the codeword.

- To ensure prefix free property, the node becomes a terminal node, thus eliminating a fraction \(D^{-s_1}\) of the terminal nodes at depth \(\ell_{\text{max}}\) (which would have been potential code words of longer length, but now they are out of the running).

- Next: chose any remaining node at level \(s_2\) (we have \((D^{s_1} - 1)D^{s_2-s_1} > 0\) choices at this point) for next codeword, thus eliminating a fraction \(D^{-s_2}\) of the nodes

- Total fraction eliminated is \(D^{-s_1} + D^{-s_2}\). 

...
Continuing this process, we eliminate a fraction \( \sum_{i=1}^{m} D^{-s_i} \) of the nodes, while retaining that the code is instantaneous (a codeword can’t be a prefix of another).


... proof of Kraft inequality cont.

- Continuing this process, we eliminate a fraction \( \sum_{i=1}^{m} D^{-s_i} \) of the nodes, while retaining that the code is instantaneous (a codeword can’t be a prefix of another).

- But since by assumption \( \sum_{i=1}^{m} D^{-s_i} \leq 1 \) we never eliminate more than all of the codewords, so this process won’t run out of codewords.
Continuing this process, we eliminate a fraction $\sum_{i=1}^{m} D^{-s_i}$ of the nodes, while retaining that the code is instantaneous (a codeword can’t be a prefix of another).

But since by assumption $\sum_{i=1}^{m} D^{-s_i} \leq 1$ we never eliminate more than all of the codewords, so this process won’t run out of codewords.

Thus, we have created a prefix-free code with the desired lengths.
**Theorem 9.3.2 (countably infinite Kraft)**

For any countably infinite set of codewords that form a prefix set, this satisfies the extended Kraft inequality, i.e.

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

(9.3)

Conversely, given $\ell_i$ satisfying the above, $\exists$ a prefix code with these lengths.

**proof of countably infinite Kraft.**

- Assume we have such a prefix code, and let the $D$-ary alphabet be $\{0, 1, \ldots, D - 1\}$. 

...
Infinite Kraft

Theorem 9.3.2 (countably infinite Kraft)

For any countably infinite set of codewords that form a prefix set, this satisfies the extended Kraft inequality, i.e.

$$\sum_{i=1}^{\infty} D^{-\ell_i} \leq 1 \tag{9.3}$$

Conversely, given \( \ell_i \) satisfying the above, \( \exists \) a prefix code with these lengths.

proof of countably infinite Kraft.

- Assume we have such a prefix code, and let the \( D \)-ary alphabet be \( \{0, 1, \ldots, D - 1\} \).
- Consider the \( i^{th} \) codeword \( y_1, y_2, \ldots, y_{\ell_i} \).
Kraft inequality

. . . proof of infinite Kraft.

- Consider expansion of codeword using fractional digits:

\[ 0.y_1y_2y_3 \ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j} \] (9.4)
Kraft inequality

... proof of infinite Kraft.

- Consider expansion of codeword using fractional digits:

\[
0.y_1 y_2 y_3 \ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j}
\] (9.4)

- Examples: When \( D = \{0, 1\} \) then \( 0.1 = 1/2, 0.01 = 1/4, 0.11 = 3/4, \) and \( 0.001 = 1/8 \) (so bits are after the binary point). 
Kraft inequality

... proof of infinite Kraft.

- Consider expansion of codeword using fractional digits:
  \[ 0.y_1y_2y_3 \ldots y_{l_i} = \sum_{j=1}^{l_i} y_j D^{-j} \]  

Examples: When \( D = \{0, 1\} \) then \( 0.1 = 1/2 \), \( 0.01 = 1/4 \), \( 0.11 = 3/4 \), and \( 0.001 = 1/8 \) (so bits are after the binary point).

- Associate each codeword \( y_1: l_i \) with the half-open interval on the real line \([0.y_1y_2 \ldots y_{l_i} , 0.y_1y_2 \ldots y_{l_i} + 1/D^{l_i})\)
Kraft inequality

... proof of infinite Kraft.

- Consider expansion of codeword using fractional digits:

\[ 0.y_1y_2y_3\ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j} \quad (9.4) \]

- Examples: When \( D = \{0, 1\} \) then \( 0.1 = 1/2 \), \( 0.01 = 1/4 \), \( 0.11 = 3/4 \), and \( 0.001 = 1/8 \) (so bits are after the binary point).

- Associate each codeword \( y_{1:\ell_i} \) with the half-open interval on the real line \([0.y_1y_2\ldots y_{\ell_i}, 0.y_1y_2\ldots y_{\ell_i} + 1/D^{\ell_i}]\)

- Example: With \( D = 10 \), then if \( 0.y_1y_2y_3 = 0.157 \), the associated half-open interval is \([0.157, 0.158)\), and if \( 0.y_1y_2y_3 = 0.159 \), the associated half-open interval is \([0.159, 0.160)\)
Kraft inequality

...proof of infinite Kraft.

- Consider expansion of codeword using fractional digits:

\[ 0.y_1y_2y_3 \ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j} \]  

(9.4)

- Examples: When \( D = \{0, 1\} \) then 0.1 = 1/2, 0.01 = 1/4, 0.11 = 3/4, and 0.001 = 1/8 (so bits are after the binary point).

- Associate each codeword \( y_{1: \ell_i} \) with the half-open interval on the real line \([0.y_1y_2 \ldots y_{\ell_i}, 0.y_1y_2 \ldots y_{\ell_i} + 1/D^{\ell_i}]\)

- Example: With \( D = 10 \), then if \( 0.y_1y_2y_3 = 0.157 \), the associated half-open interval is \([0.157, 0.158)\), and if \( 0.y_1y_2y_3 = 0.159 \), the associated half-open interval is \([0.159, 0.160)\)
Kraft inequality

...proof of infinite Kraft.

- So the interval for codeword $y_1 y_2 y_3 \ldots y_\ell_i$ corresponds to the set of all real numbers that begins with $0.y_1 y_2 y_3 \ldots y_\ell_i$. 

Length of interval for codeword $y_1 y_2 y_3 \ldots y_\ell_i$ is $D_\ell_i$.

And since all intervals live in $[0, 1)$ we must have $\sum_{i=1}^{\infty} D_\ell_i \leq 1$. 

Proof of converse is similar to finite case and also to arithmetic coding that we’ll soon see, so we skip the proof here.
So the interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ corresponds to the set of all real numbers that begins with $0.y_1y_2y_3 \ldots y_{\ell_i}$ and is thus a sub-interval of the unit interval.
So the interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ corresponds to the set of all real numbers that begins with $0.y_1y_2y_3 \ldots y_{\ell_i}$ and is thus a sub-interval of the unit interval.

Also $y_1y_2y_3 \ldots y_{\ell_i}$ is not a prefix of any other codeword, so the intervals must be disjoint.
Kraft inequality

...proof of infinite Kraft.

- So the interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ corresponds to the set of all real numbers that begins with $0.y_1y_2y_3 \ldots y_{\ell_i}$ and is thus a sub-interval of the unit interval.
- Also $y_1y_2y_3 \ldots y_{\ell_i}$ is not a prefix of any other codeword, so the intervals must be disjoint.
- Length of interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ is $D^{-\ell_i}$. 
So the interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ corresponds to the set of all real numbers that begins with $0.y_1y_2y_3 \ldots y_{\ell_i}$ and is thus a sub-interval of the unit interval.

Also $y_1y_2y_3 \ldots y_{\ell_i}$ is not a prefix of any other codeword, so the intervals must be disjoint.

Length of interval for codeword $y_1y_2y_3 \ldots y_{\ell_i}$ is $D^{-\ell_i}$.

And since all intervals live in $[0, 1)$ we must have

$$\sum_i D^{-\ell_i} \leq 1 \quad (9.5)$$
...proof of infinite Kraft.

- So the interval for codeword $y_1 y_2 y_3 \ldots y_{\ell_i}$ corresponds to the set of all real numbers that begins with $0.y_1 y_2 y_3 \ldots y_{\ell_i}$ and is thus a sub-interval of the unit interval.
- Also $y_1 y_2 y_3 \ldots y_{\ell_i}$ is not a prefix of any other codeword, so the intervals must be disjoint.
- Length of interval for codeword $y_1 y_2 y_3 \ldots y_{\ell_i}$ is $D^{-\ell_i}$.
- And since all intervals live in $[0, 1)$ we must have

$$\sum_i D^{-\ell_i} \leq 1 \quad (9.5)$$

- Proof of converse is similar to finite case and also to arithmetic coding that we’ll soon see, so we skip the proof here.
Towards Optimal Codes

- Summarizing: Prefix code $\Leftrightarrow$ Kraft inequality.
Towards Optimal Codes

- Summarizing: Prefix code ⇔ Kraft inequality.
- Thus, we need only find lengths that satisfy Kraft to find a prefix code.
Towards Optimal Codes

- Summarizing: Prefix code $\iff$ Kraft inequality.
- Thus, we need only find lengths that satisfy Kraft to find a prefix code.
- Goal: find a prefix code with minimum expected length

$$L(C) = \sum_{i} p_i l_i \quad (9.6)$$
Towards Optimal Codes

- Summarizing: Prefix code $\Leftrightarrow$ Kraft inequality.
- Thus, we need only find lengths that satisfy Kraft to find a prefix code.
- Goal: find a prefix code with minimum expected length

$$L(C) = \sum_i p_i \ell_i$$  \hspace{1cm} (9.6)

- This is a constrained optimization problem:

$$\text{minimize} \quad \sum_{\ell_1,m} \sum_i p_i \ell_i$$

$$\text{subject to} \quad \sum_i D^{-\ell_i} \leq 1$$  \hspace{1cm} (9.7)
Towards Optimal Codes

- Summarizing: Prefix code $\Leftrightarrow$ Kraft inequality.
- Thus, we need only find lengths that satisfy Kraft to find a prefix code.
- Goal: find a prefix code with minimum expected length

$$L(C) = \sum_i p_i \ell_i$$ (9.6)

- This is a constrained optimization problem:

$$\min_{\{\ell_1:m\} \in \mathbb{Z}_+^m} \sum_i p_i \ell_i$$ (9.7)

$$\text{subject to } \sum_i D^{-\ell_i} \leq 1$$

- Integer program is an NP-complete optimization, not likely to be efficiently solvable (unless $P=NP$).
Towards Optimal Codes

- Relax the integer constraints on $\ell_i$ for now, and consider Lagrangian

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)$$  \hspace{1cm} (9.8)
Towards Optimal Codes

- Relax the integer constraints on $\ell_i$ for now, and consider Lagrangian

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)$$  \hspace{1cm} (9.8)

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = \frac{1}{\ln D} \hspace{1cm} (9.10)$$

$$\ell_i = \log_2 D p_i \hspace{1cm} (9.12)$$
Towards Optimal Codes

Relax the integer constraints on $\ell_i$ for now, and consider the Lagrangian

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)$$  \hspace{1cm} (9.8)

Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0$$  \hspace{1cm} (9.9)

$$D^{-\ell} = e^{\ln b^{-\ell}} = \ldots$$  \hspace{1cm} (9.12)
Towards Optimal Codes

- Relax the integer constraints on $\ell_i$ for now, and consider Lagrangian

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right) \quad (9.8)$$

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 \quad (9.9)$$

$$\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} \quad (9.10)$$

$$\Rightarrow \ell_i = \log_D \frac{p_i}{\lambda} \quad (9.12)$$
Towards Optimal Codes

- Relax the integer constraints on \( \ell_i \) for now, and consider Lagrangian

\[
J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)
\]  

(9.8)

- Taking derivatives and setting to 0,

\[
\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0
\]  

(9.9)

\[
\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D}
\]  

(9.10)

\[
\frac{\partial J}{\partial \lambda}
\]  

(9.12)
Towards Optimal Codes

Relax the integer constraints on $\ell_i$ for now, and consider Lagrangian

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)$$  \hspace{1cm} (9.8)

Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0$$ \hspace{1cm} (9.9)

$$\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D}$$ \hspace{1cm} (9.10)

$$\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1$$ \hspace{1cm} (9.12)
Towards Optimal Codes

- Relax the integer constraints on \( \ell_i \) for now, and consider Lagrangian

\[
J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right) \tag{9.8}
\]

- Taking derivatives and setting to 0,

\[
\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 \tag{9.9}
\]

\[
\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} \tag{9.10}
\]

\[
\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0 \tag{9.12}
\]
Towards Optimal Codes

- Relax the integer constraints on $\ell_i$ for now, and consider Lagrangian

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right) \quad (9.8)$$

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 \quad (9.9)$$

$$\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} \quad (9.10)$$

$$\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0 \quad \Rightarrow \quad \lambda = 1 / \ln D \quad (9.11)$$

$$\quad \Rightarrow \quad \lambda = 1 / \ln D \quad (9.12)$$
Towards Optimal Codes

Relax the integer constraints on $\ell_i$ for now, and consider Lagrangian

\[
J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right) \tag{9.8}
\]

Taking derivatives and setting to 0,

\[
\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 \tag{9.9}
\]

\[\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} \tag{9.10}\]

\[
\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0 \quad \Rightarrow \quad \lambda = 1 / \ln D \tag{9.11}
\]

\[\Rightarrow D^{-\ell_i} = p_i \tag{9.12}\]
Towards Optimal Codes

- Relax the integer constraints on \( \ell_i \) for now, and consider Lagrangian

\[
J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right)
\]  
(9.8)

- Taking derivatives and setting to 0,

\[
\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0
\]  
(9.9)

\[
\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D}
\]  
(9.10)

\[
\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0 \quad \Rightarrow \quad \lambda = \frac{1}{\ln D}
\]  
(9.11)

\[
\Rightarrow D^{-\ell_i} = p_i \quad \text{yielding} \quad \ell_i^* = -\log_D p_i
\]  
(9.12)
Towards Optimal Codes

- This implies that:

$$L^*$$  \hspace{1cm} (9.13)
Towards Optimal Codes

This implies that:

\[ L^* = \sum_i p_i \ell_i \]  

(9.13)
Towards Optimal Codes

This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i \]

(9.13)
Towards Optimal Codes

This implies that:

\[ L^* = \sum_i p_i \ell_i^* = -\sum_i p_i \log_D p_i = H_D(X) \]  

(9.13)
Towards Optimal Codes

- This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = H(X) / \log D \quad (9.13) \]
Towards Optimal Codes

- This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = \frac{H(X)}{\log D} \quad (9.13) \]

- So the optimal expected code length, as a result of this optimization process, is the entropy
Towards Optimal Codes

- This implies that:

\[ L^* = \sum_i p_i \ell_i^* = -\sum_i p_i \log_D p_i = H_D(X) = H(X) / \log D \quad (9.13) \]

- So the optimal expected code length, as a result of this optimization process, is the entropy assuming that we are allowed to have fractional code lengths.
Towards Optimal Codes

- This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = H(X)/\log D \quad (9.13) \]

- So the optimal expected code length, as a result of this optimization process, is the entropy assuming that we are allowed to have fractional code lengths.

- Since \( \ell_i^* = - \log_D p_i \), this means that optimal code “length” (while fractional) is the same as the information about the event. I.e., shortest possible coding length is the inherent information about an event. This is like the MDL (minimum description principle), tries to find the simplest explanation about a source.
Towards Optimal Codes

- This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = H(X) / \log D \quad (9.13) \]

- So the optimal expected code length, as a result of this optimization process, is the entropy assuming that we are allowed to have fractional code lengths.

- Since \( \ell_i^* = - \log_D p_i \), this means that optimal code “length” (while fractional) is the same as the information about the event. I.e., shortest possible coding length is the inherent information about an event. This is like the MDL (minimum description principle), tries to find the simplest explanation about a source.

- Compare fractional codeword lengths to long block codes, what is the relation?
Theorem 9.3.3

Entropy is the minimum expected length. That is, the expected length $L$ of any instantaneous $D$-ary code (which thus satisfies Kraft inequality) for a r.v. $X$ is such that

$$L \geq H_D(X)$$

(9.14)

with equality iff $D^{-\ell_i} = p_i$. 
Proof of Theorem 9.3.3.

\[ L - H_D(X) \]  
(9.15)

\[ \text{(9.17)} \]
\[ \text{(9.18)} \]
\[ \text{(9.20)} \]

\[ \ldots \]
Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i} \quad (9.15) \]

\[ \log D \left( \sum_i p_i \ell_i \right) = \sum_i p_i \log_D \frac{1}{p_i} \quad (9.17) \]

\[ \log D (\sum_i p_i \ell_i) = \sum_i p_i \log_D \frac{1}{p_i} \quad (9.18) \]

\[ \log D (1/c) \quad (9.20) \]
Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i} \]  \hspace{1cm} (9.15)

\[ = -\sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \]  \hspace{1cm} (9.16)

\[ \text{since} \quad c \leq 1 \text{ by Kraft, where} \quad c = \sum_i D^{-\ell_i} \]  \hspace{1cm} (9.20)
Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D 1/p_i \]  
(9.15)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \]  
(9.16)

\[ = - \sum_i p_i \log_D D^{-\ell_i} \]  
(9.17)

\[ + \sum_i p_i \log_D p_i \]  
(9.18)

(9.20)

...
Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i} \]  
(9.15)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \]  
(9.16)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \log_D (\sum_i D^{-\ell_i}) - \log_D (\sum_i D^{-\ell_i}) \]  
(9.17)

\[ + \sum_i p_i \log_D p_i \]  
(9.18)

(9.20)

...
Optimal Code Lengths

Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D 1/p_i \] (9.15)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \] (9.16)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \log_D \left( \sum_i D^{-\ell_i} \right) - \log_D \left( \sum_i D^{-\ell_i} \right) \] (9.17)

\[ + \sum_i p_i \log_D p_i \quad \text{(now define } r_i = \frac{D^{-\ell_i}}{\sum_i D^{-\ell_i}}) \] (9.18)

(9.20)

...
Optimal Code Lengths

Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i} \]  \hspace{2cm} (9.15)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \]  \hspace{2cm} (9.16)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \log_D \left( \sum_i D^{-\ell_i} \right) - \log_D \left( \sum_i D^{-\ell_i} \right) \]  \hspace{2cm} (9.17)

\[ + \sum_i p_i \log_D p_i \quad \text{(now define } r_i = \frac{D^{-\ell_i}}{\sum_i D^{-\ell_i}}) \]  \hspace{2cm} (9.18)

\[ = \sum_i p_i \log \frac{p_i}{r_i} - \log_D \left( \sum_i D^{-\ell_i} \right) \]  \hspace{2cm} (9.20)

...
Proof of Theorem 9.3.3.

\[ L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D 1/p_i \]  \hfill (9.15)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \]  \hfill (9.16)

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \log_D \left( \sum_i D^{-\ell_i} \right) - \log_D \left( \sum_i D^{-\ell_i} \right) \]  \hfill (9.17)

\[ + \sum_i p_i \log_D p_i \quad \text{ (now define } r_i = \frac{D^{-\ell_i}}{\sum_i D^{-\ell_i}} \text{)} \]  \hfill (9.18)

\[ = \sum_i p_i \log \frac{p_i}{r_i} - \log_D \left( \sum_i D^{-\ell_i} \right) = D(p||r) + \log_D (1/c) \]  \hfill (9.19)

\[ C = - \mathbb{E} 0^{-\ell_i} \geq 0 \quad \text{ } \geq 0 \]  \hfill (9.20)

...
Proof of Theorem 9.3.3.

\[ L = H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D 1/p_i \]  

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i \]  

\[ = - \sum_i p_i \log_D D^{-\ell_i} + \log_D \left( \sum_i D^{-\ell_i} \right) - \log_D \left( \sum_i D^{-\ell_i} \right) \]  

\[ + \sum_i p_i \log_D p_i \quad \text{(now define } r_i = \frac{D^{-\ell_i}}{\sum_i D^{-\ell_i}} ) \]  

\[ = \sum_i p_i \log \frac{p_i}{r_i} - \log_D \left( \sum_i D^{-\ell_i} \right) = D(p|\|r) + \log_D(1/c) \]  

\[ \geq 0 \quad \text{since } c \leq 1 \text{ by Kraft, where } c = \sum_i D^{-\ell_i} \]
...Proof of Theorem 9.3.3.

- So we have that $L \geq H_D(X)$. 

Equality, $L = H_D$ is achieved iff $p_i = D_i$ for all $i$, $\log D_i p_i$ is an integer ...in which case $c = P_i D_i = 1$
Proof of Theorem 9.3.3.

- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \iff -\log_D p_i$ is an integer . . .
...Proof of Theorem 9.3.3.

- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \Leftrightarrow -\log_D p_i$ is an integer ...
- ...in which case $c = \sum_i D^{-\ell_i} = 1$
Optimal Code Lengths

...Proof of Theorem 9.3.3.

- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \iff -\log_D p_i$ is an integer ...
- ...in which case $c = \sum_i D^{-\ell_i} = 1$
Optimal Code Lengths

...Proof of Theorem 9.3.3.

- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \Leftrightarrow -\log_D p_i$ is an integer ...
- ...in which case $c = \sum_i D^{-\ell_i} = 1$

Definition 9.3.4 ($D$-adic)

A probability distribution is called $D$-adic w.r.t. $D$ if each of the probabilities is $= D^{-n}$ for some $n$. 
Optimal Code Lengths

...Proof of Theorem 9.3.3.

- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \iff -\log_D p_i$ is an integer ...
- ... in which case $c = \sum_i D^{-\ell_i} = 1$

**Definition 9.3.4 ($D$-adic)**

A probability distribution is called $D$-adic w.r.t. $D$ if each of the probabilities is $= D^{-n}$ for some $n$.

- Ex: when $D = 2$, the distribution $[\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}] = [2^{-1}, 2^{-2}, 2^{-3}, 2^{-3}]$ is 2-adic.
**Optimal Code Lengths**

...Proof of Theorem 9.3.3.
- So we have that $L \geq H_D(X)$.
- Equality, $L = H$ is achieved iff $p_i = D^{-\ell_i}$ for all $i \iff -\log_D p_i$ is an integer ...
- ...in which case $c = \sum_i D^{-\ell_i} = 1$

**Definition 9.3.4 ($D$-adic)**

A probability distribution is called $D$-adic w.r.t. $D$ if each of the probabilities is $= D^{-n}$ for some $n$.

- Ex: when $D = 2$, the distribution $[\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}] = [2^{-1}, 2^{-2}, 2^{-3}, 2^{-3}]$ is 2-adic.
- Thus, we have equality above iff the distribution is appropriately $D$-adic.
Summary of recent results

To summarize the conditions and relations:

- In the relaxed optimization problem, we relaxed the lengths so that they need not be integers, and we get $E\ell = H$.
Summary of recent results

To summarize the conditions and relations:

- In the relaxed optimization problem, we relaxed the lengths so that they need not be integers, and we get \( \mathbb{E}\ell = H \).
- Assuming Kraft is true (and thus a prefix code exists), we have (with integer lengths) that \( \mathbb{E}\ell \geq H \).
Summary of recent results

To summarize the conditions and relations:

- In the relaxed optimization problem, we relaxed the lengths so that they need not be integers, and we get $E \ell = H$.
- Assuming Kraft is true (and thus a prefix code exists), we have (with integer lengths) that $E \ell \geq H$.
- I.e., if we assume Kraft, and $\ell_i = -\log D p_i$ is an integer, then $E \ell = H$.
Summary of recent results

To summarize the conditions and relations:

- In the relaxed optimization problem, we relaxed the lengths so that they need not be integers, and we get $E\ell = H$.
- Assuming Kraft is true (and thus a prefix code exists), we have (with integer lengths) that $E\ell \geq H$.
- I.e., if we assume Kraft, and $\ell_i = -\log_D p_i$ is an integer, then $E\ell = H$.
- I.e., if we assume Kraft, and $\ell_i \neq -\log_D p_i$, but the lengths $\ell_i$ are still integers, then we have $E\ell > H$ strictly.
Shannon Codes

- \[ L - H = D(p||r) + \log_D 1/c, \text{ with } c = \sum_i D^{-\ell_i} \]
Shannon Codes

- \( L - H = D(p|r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)

- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.
Shannon Codes

- \( L - H = D(p|r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)

Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.

In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).
Shannon Codes

- \( L - H = D(p|r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)
- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.
- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).
- **Shannon codes**: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths
Shannon Codes

- \( L - H = D(p||r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)
- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.
- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).
- **Shannon codes**: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths
- \( \sum_i D^{-\ell_i} \)
Shannon Codes

- \( L - H = D(p|r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)

Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.

In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).

- **Shannon codes**: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths

\[
\sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil}
\]
Shannon Codes

- \( L - H = D(p||r) + \log_D \frac{1}{c}, \) with \( c = \sum_i D^{-\ell_i} \)

Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.

In general, however, unless \( \text{P=NP} \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).

**Shannon codes:** consider \( \ell_i = \lceil \log_D \frac{1}{p_i} \rceil \) as the code lengths

\[
\sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil} \leq \sum_i D^{-\log 1/p_i}
\]
Shannon Codes

- $L - H = D(p||r) + \log_D 1/c$, with $c = \sum_i D^{-\ell_i}$

Thus, to produce a code, we find closest (in the KL sense) $D$-adic distribution w.r.t. $D$ to $p$ and then construct the code as in the proof of the Kraft inequality converse.

In general, however, unless $P=NP$, it is hard to find the KL closest $D$-adic distribution (integer programming problem).

**Shannon codes:** consider $\ell_i = \lceil \log_D 1/p_i \rceil$ as the code lengths

- $\sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil} \leq \sum_i D^{-\log 1/p_i} = \sum_i p_i$
Shannon Codes

- \( L - H = D(p||r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)
- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.
- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).
- Shannon codes: consider \( \ell_i = \lfloor \log_D 1/p_i \rfloor \) as the code lengths
- \( \sum_i D^{-\ell_i} = \sum_i D^{-\lfloor \log 1/p_i \rfloor} \leq \sum_i D^{-\log 1/p_i} = \sum_i p_i = 1 \)
Shannon Codes

- \( L - H = D(p|\pi) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)
- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.
- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).
- **Shannon codes**: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths
- \( \sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil} \leq \sum_i D^{-\log 1/p_i} = \sum_i p_i = 1 \)
- This means Kraft inequality holds for these lengths, so there is a prefix code
Shannon Codes

- \( L - H = D(p|r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)

- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.

- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).

- **Shannon codes**: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths

  \[
  \sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil} \leq \sum_i D^{-\log 1/p_i} = \sum_i p_i = 1
  \]

- This means Kraft inequality holds for these lengths, so there is a prefix code (if the lengths were too short there might be a problem but we’re rounding up).
Shannon Codes

- \( L - H = D(p|r) + \log_D 1/c \), with \( c = \sum_i D^{-\ell_i} \)
- Thus, to produce a code, we find closest (in the KL sense) \( D \)-adic distribution w.r.t. \( D \) to \( p \) and then construct the code as in the proof of the Kraft inequality converse.
- In general, however, unless \( P=NP \), it is hard to find the KL closest \( D \)-adic distribution (integer programming problem).
- Shannon codes: consider \( \ell_i = \lceil \log_D 1/p_i \rceil \) as the code lengths
- \( \sum_i D^{-\ell_i} = \sum_i D^{-\lceil \log 1/p_i \rceil} \leq \sum_i D^{-\log 1/p_i} = \sum_i p_i = 1 \)
- This means Kraft inequality holds for these lengths, so there is a prefix code (if the lengths were too short there might be a problem but we’re rounding up).
- Also, we have a bound on lengths in terms of real numbers

\[
\log_D \frac{1}{p_i} \leq \ell_i < \log_D \frac{1}{p_i} + 1 \quad (9.21)
\]
Shannon Codes

- Taking expected values on both sides yields

\[ H_D(X) \leq L < H_D(X) + 1 \]  \hspace{1cm} (9.22)
Shannon Codes

- Taking expected values on both sides yields

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

(9.22)

\[ \Rightarrow \Rightarrow \]

- Close to the entropy, only one extra bit!
Shannon Codes

- Taking expected values on both sides yields

\[ H_D(X) \leq L < H_D(X) + 1 \]  \hspace{1cm} (9.22)

- Close to the entropy, only one extra bit! Is this good?
Shannon Codes

- Taking expected values on both sides yields

\[ H_D(X) \leq L < H_D(X) + 1 \]  \hspace{1cm} (9.22)

- Close to the entropy, only one extra bit! Is this good?
- Also, \( H \leq L^* \leq L \) where \( L^* \) is the optimal length for integer length codes (i.e., might not have optimal integer lengths satisfying Kraft).
Shannon Codes

- Taking expected values on both sides yields

\[ H_D(X) \leq L < H_D(X) + 1 \] (9.22)

- Close to the entropy, only one extra bit! Is this good?
- Also, \( H \leq L^* \leq L \) where \( L^* \) is the optimal length for integer length codes (i.e., might not have optimal integer lengths satisfying Kraft).

\[ H \leq L^* \leq L < H + 1 \]
Shannon Codes

- Taking expected values on both sides yields
  \[ H_D(X) \leq L < H_D(X) + 1 \]  
  (9.22)

- Close to the entropy, only one extra bit! Is this good?
- Also, \( H \leq L^* \leq L \) where \( L^* \) is the optimal length for integer length codes (i.e., might not have optimal integer lengths satisfying Kraft).

**Theorem 9.4.1**

Let \( \ell_1^*, \ell_2^*, \ldots, \ell_m^* \) be the optimal integral codeword lengths for source \( p \) and \( D \)-ary alphabet. \( L^* \) is the expected length. Then

\[ H_D(X) \leq L^* < H_D(X) + 1 \]  
(9.23)
Shannon Codes

- Taking expected values on both sides yields

\[ H_D(X) \leq L < H_D(X) + 1 \quad (9.22) \]

- Close to the entropy, only one extra bit! Is this good?
- Also, \( H \leq L^* \leq L \) where \( L^* \) is the optimal length for integer length codes (i.e., might not have optimal integer lengths satisfying Kraft).

**Theorem 9.4.1**

Let \( \ell_1^*, \ell_2^*, \ldots, \ell_m^* \) be the optimal integral codeword lengths for source \( p \) and \( D \)-ary alphabet. \( L^* \) is the expected length. Then

\[ H_D(X) \leq L^* < H_D(X) + 1 \quad (9.23) \]

- So average overhead of using integers (rather than fractional) codeword lengths is no more than one bit per symbol.
How bad is one bit?

- How bad is this overhead?

If \( E^*(X) = H_D(X) + 1 \), then the efficiency is \( < 1 \). Efficiency is \( > 0 \) as \( H(X) > 0 \), so entropy would need to be very large for this to be good. For small alphabets (or low-entropy distributions, such as close to deterministic distributions), impossible to have good efficiency.

E.g., \( D = \{0, 1\} \) then \( \max H(X) = 1 \), so best possible efficiency is 50%.
How bad is one bit?

- How bad is this overhead?
- Depends on $H$. Efficiency of code

$$0 \leq \text{Efficiency} \triangleq \frac{H_D(X)}{E\ell(X)} \leq 1$$ (9.24)
How bad is one bit?

- How bad is this overhead?
- Depends on $H$. Efficiency of code

\[
0 \leq \text{Efficiency} \triangleq \frac{H_D(X)}{E\ell(X)} \leq 1
\]  \hspace{1cm} (9.24)

- If $E\ell(X) = H_D(X) + 1$, then efficiency $\to 1$ as $H(X) \to \infty$. 
How bad is one bit?

- How bad is this overhead?
- Depends on $H$. Efficiency of code

$$0 \leq \text{Efficiency} \triangleq \frac{H_D(X)}{E\ell(X)} \leq 1$$  \hspace{1cm} (9.24)

- If $E\ell(X) = H_D(X) + 1$, then efficiency $\to 1$ as $H(X) \to \infty$.
- Efficiency $\to 0$ as $H(X) \to 0$, so entropy would need to be very large for this to be good.
How bad is one bit?

- How bad is this overhead?
- Depends on $H$. Efficiency of code

$$0 \leq \text{Efficiency} \triangleq \frac{H_D(X)}{E\ell(X)} \leq 1 \quad (9.24)$$

- If $E\ell(X) = H_D(X) + 1$, then efficiency $\to 1$ as $H(X) \to \infty$.
- Efficiency $\to 0$ as $H(X) \to 0$, so entropy would need to be very large for this to be good.
- For small alphabets (or low-entropy distributions, such as close to deterministic distributions), impossible to have good efficiency.
How bad is one bit?

- How bad is this overhead?
- Depends on $H$. Efficiency of code

\[
0 \leq \text{Efficiency} \triangleq \frac{H_D(X)}{E\ell(X)} \leq 1
\]  

(9.24)

- If $E\ell(X) = H_D(X) + 1$, then efficiency $\to 1$ as $H(X) \to \infty$.
- Efficiency $\to 0$ as $H(X) \to 0$, so entropy would need to be very large for this to be good.
- For small alphabets (or low-entropy distributions, such as close to deterministic distributions), impossible to have good efficiency. E.g., $\mathcal{D} = \{0, 1\}$ then $\max H(X) = 1$, so best possible efficiency is 50%.
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $> 1$ symbol at a time (block code, or a vector code, the symbol is the vector).
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $> 1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected per-symbol length for $n$ symbols $x_{1:n}$.
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $> 1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected per-symbol length for $n$ symbols $x_{1:n}$. $L_n$ is the expected per-symbol length, when encoding $n$ symbols.
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $>1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected per-symbol length for $n$ symbols $x_{1:n}$. $L_n$ is the expected per-symbol length, when encoding $n$ symbols.

$$L_n = \frac{1}{n} \sum_{x_{1:n}} p(x_{1:n}) \ell(x_{1:n}) = \frac{1}{n} E\ell(x_{1:n})$$  \hspace{5cm} (9.25)
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $> 1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected per-symbol length for $n$ symbols $x_{1:n}$. $L_n$ is the expected per-symbol length, when encoding $n$ symbols.

\[
L_n = \frac{1}{n} \sum_{x_{1:n}} p(x_{1:n}) \ell(x_{1:n}) = \frac{1}{n} E\ell(x_{1:n}) \quad (9.25)
\]

- Let's use Shannon coding lengths to get

\[
\log \frac{1}{p_i} \leq \ell_i \leq \log \frac{1}{p_i} + 1 \quad (9.26)
\]

\[
(9.27)
\]
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $> 1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected per-symbol length for $n$ symbols $x_{1:n}$. $L_n$ is the expected per-symbol length, when encoding $n$ symbols.

$$L_n = \frac{1}{n} \sum_{x_{1:n}} p(x_{1:n}) \ell(x_{1:n}) = \frac{1}{n} E\ell(x_{1:n})$$  \hspace{1cm} (9.25)

- Let's use Shannon coding lengths to get

$$\sum_i p_i \left( \log \frac{1}{p_i} \leq \ell_i \leq \log \frac{1}{p_i} + 1 \right)$$  \hspace{1cm} (9.26)

$$H(X_1, ..., X_n) \leq E\ell(X_{1:n}) + 1$$  \hspace{1cm} (9.27)
Improving efficiency

- Such symbol codes are inherently disadvantaged, unless their distributions are $D$-adic.
- We can reduce overhead (improve efficiency) by coding $>1$ symbol at a time (block code, or a vector code, the symbol is the vector).
- Let $L_n$ be the expected per-symbol length for $n$ symbols $x_{1:n}$. $L_n$ is the expected per-symbol length, when encoding $n$ symbols.

\[
L_n = \frac{1}{n} \sum_{x_{1:n}} p(x_{1:n}) \ell(x_{1:n}) = \frac{1}{n} E\ell(x_{1:n}) \tag{9.25}
\]

- Let's use Shannon coding lengths to get

\[
\sum_i p_i \left( \log \frac{1}{p_i} \leq \ell_i \leq \log \frac{1}{p_i} + 1 \right) \tag{9.26}
\]

\[
\Rightarrow H(X_1, \ldots, X_n) \leq E\ell(x_{1:n}) < H(X_1, \ldots, X_n) + 1 \tag{9.27}
\]
Improving efficiency

- If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$. 
If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$.

$\Rightarrow$ we have that

$$H(X) \leq L_n \leq H(X) + \frac{1}{n} \quad (9.28)$$
Improving efficiency

- If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$.
- $\Rightarrow$ we have that

$$H(X) \leq L_n \leq H(X) + \frac{1}{n} \tag{9.28}$$

- As $n$ gets big, per symbol penalty of a Shannon code decreases,
If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$.

⇒ we have that

$$H(X) \leq L_n \leq H(X) + \frac{1}{n} \quad (9.28)$$

As $n$ gets big, per symbol penalty of a Shannon code decreases, and we approach the Entropy limit (per symbol),
If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$.

- If $X_1, \ldots, X_n$ are i.i.d., then $H(X_1, \ldots, X_n) = nH(X_i)$.
- If $X_1, \ldots, X_n$ are i.i.d., then $H(X_1, \ldots, X_n) = nH(X_i)$.

\[ H(X) \leq L_n \leq H(X) + \frac{1}{n} \] (9.28)

As $n$ gets big, per symbol penalty of a Shannon code decreases, and we approach the Entropy limit (per symbol), although once again we have to code a block at a time.
Improving efficiency

- If the $X_i$ are i.i.d. then $H(X_1, \ldots, X_n) = nH(X_i)$.
- $\Rightarrow$ we have that
  $$H(X) \leq L_n \leq H(X) + \frac{1}{n} \quad (9.28)$$
- As $n$ gets big, per symbol penalty of a Shannon code decreases, and we approach the Entropy limit (per symbol), although once again we have to code a block at a time.
- Again, even if symbols are independent it is better to code jointly.
Stochastic processes

Consider any stationary (ergodic) stochastic process. Then

\[ H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1 \]  \hspace{1cm} (9.29)

\[ H(X_1, \ldots, X_n) \leq L^* \leq H(X_1, \ldots, X_n) + 1 \]  \hspace{1cm} (9.30)
Consider any stationary (ergodic) stochastic process. Then

\[ H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1 \quad (9.29) \]

\[ \Rightarrow \frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n} \quad (9.30) \]
Stochastic processes

- Consider any stationary (ergodic) stochastic process. Then

\[
H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1 \quad (9.29)
\]

\[
\Rightarrow \frac{H(X_1, \ldots, X_n)}{n} \leq \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n} \quad (9.30)
\]

- If stationary, then l.h.s. \( \rightarrow H(\mathcal{X}) \) as \( n \rightarrow \infty \).
Stochastic processes

Consider any stationary (ergodic) stochastic process. Then

$$H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1$$  \hspace{1cm} (9.29)

$$\Rightarrow \frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}$$  \hspace{1cm} (9.30)

If stationary, then l.h.s. $\rightarrow H(\mathcal{X})$ as $n \rightarrow \infty$.

Thus, as $n$ gets large, expected length of code goes to the entropy rate of the stochastic process.
Stochastic processes

- Consider any stationary (ergodic) stochastic process. Then

\[ H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1 \] (9.29)

\[ \Rightarrow \frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n} \] (9.30)

- If stationary, then l.h.s. → \( H(\mathcal{X}) \) as \( n \to \infty \).
- Thus, as \( n \) gets large, expected length of code goes to the entropy rate of the stochastic process.
- We can make penalty per source symbol as small as we want if we don’t mind long block lengths. This can be stated as a theorem
Consider any stationary (ergodic) stochastic process. Then

$$H(X_1, \ldots, X_n) \leq E\ell(X_1, \ldots, X_n) < H(X_1, \ldots, X_n) + 1$$  \hspace{1cm} (9.29)

$$\Rightarrow \frac{H(X_1, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}$$  \hspace{1cm} (9.30)

If stationary, then l.h.s. $\to H(X)$ as $n \to \infty$.

Thus, as $n$ gets large, expected length of code goes to the entropy rate of the stochastic process.

We can make penalty per source symbol as small as we want if we don’t mind long block lengths. This can be stated as a theorem

**Theorem 9.4.2**

*Minimum expected codeword lengths per symbol satisfy*

$$\frac{H(X_1, \ldots, X_n)}{n} \leq L^*_n < \frac{H(X_1, \ldots, X_n)}{n} + \frac{1}{n}$$  \hspace{1cm} (9.31)

*if $X_i$ is stationary. I.e., $L^* \to H(X)$*
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes.

\[
\begin{align*}
\text{(9.35)}
\end{align*}
\]
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. i.e., Shannon code would use lengths \( \ell(x) = \lceil \log 1/q(x) \rceil \) but the true probability is \( p(x) \neq q(x) \).

\[
D(p || q) + H(p) + 1 \tag{9.35}
\]
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths \( \ell(x) = \lceil \log 1/q(x) \rceil \) but the true probability is \( p(x) \neq q(x) \). How does this hurt us?
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths \( \ell(x) = \lceil \log 1/q(x) \rceil \) but the true probability is \( p(x) \neq q(x) \). How does this hurt us?

\[
E\ell(X)
\]

(9.35)
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths \( \ell(x) = \lfloor \log 1/q(x) \rfloor \) but the true probability is \( p(x) \neq q(x) \). How does this hurt us?

\[
E\ell(X) = \sum_x p(x) \lfloor \log 1/q(x) \rfloor
\]

(9.35)
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths $\ell(x) = \lceil \log \frac{1}{q(x)} \rceil$ but the true probability is $p(x) \neq q(x)$. How does this hurt us?

$$E\ell(X) = \sum_x p(x) \lceil \log \frac{1}{q(x)} \rceil \leq \sum_x p(x) \left( \log \frac{1}{q(x)} + 1 \right)$$  \hspace{1cm} (9.32)

$$D(p || q) + H(p) + 1$$  \hspace{1cm} (9.35)
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths $\ell(x) = \lceil \log \frac{1}{q(x)} \rceil$ but the true probability is $p(x) \neq q(x)$. How does this hurt us?

\[
E\ell(X) = \sum_x p(x) \lceil \log \frac{1}{q(x)} \rceil \leq \sum_x p(x) (\log \frac{1}{q(x)} + 1) \quad (9.32)
\]

\[
= \sum_x p(x) (\log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1) \quad (9.33)
\]

\[
= \sum_x p(x) \log \frac{1}{q(x)} + 1 \quad (9.35)
\]
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths $\ell(x) = \lceil \log 1/q(x) \rceil$ but the true probability is $p(x) \neq q(x)$. How does this hurt us?

$$E\ell(X) = \sum_x p(x) \lceil \log 1/q(x) \rceil \leq \sum_x p(x) (\log \frac{1}{q(x)} + 1)$$ (9.32)

$$= \sum_x p(x) (\log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1)$$ (9.33)

$$= \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1$$ (9.34)

$$\text{Thus, } D(p||q) + H(p) + 1$$ (9.35)
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths $\ell(x) = \lceil \log 1/q(x) \rceil$ but the true probability is $p(x) \neq q(x)$. How does this hurt us?

\[
E\ell(X) = \sum_x p(x) \lceil \log 1/q(x) \rceil \leq \sum_x p(x) (\log \frac{1}{q(x)} + 1) \tag{9.32}
\]

\[
= \sum_x p(x) (\log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1) \tag{9.33}
\]

\[
= \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \tag{9.34}
\]

\[
= D(p||q) + H(p) + 1 \tag{9.35}
\]
Coding with the wrong distribution

- In general, we don’t have the “true” distribution (if there is one).
- With the wrong distribution, we’ll make mistakes. I.e., Shannon code would use lengths $\ell(x) = \lceil \log 1/q(x) \rceil$ but the true probability is $p(x) \neq q(x)$. How does this hurt us?

\[
\frac{1}{n} E\ell(X) = \sum_x p(x) \lceil \log 1/q(x) \rceil \leq \sum_x p(x) \left( \log \frac{1}{q(x)} + 1 \right) \tag{9.32}
\]

\[
= \sum_x p(x) \left( \log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1 \right) \tag{9.33}
\]

\[
= \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \tag{9.34}
\]

\[
= D(p\|q) + H(p) + 1 \tag{9.35}
\]

- Thus, $D(p\|q)$ is per symbol bit penalty for using wrong distribution.
Coding with the wrong distribution

**Theorem 9.4.3**

*Expected length under* $p(x)$ *of code with* $\ell(x) = \lceil \log \frac{1}{q(x)} \rceil$ *satisfies*

$$H(p) + D(p||q) \leq E_p\ell(X) \leq H(p) + D(p||q) + 1 \quad (9.36)$$

- l.h.s. is the best we can do with the wrong distribution $q$ when the true distribution is $p$. 
Goal is to find a code with the shortest possible expected length.

From the above code class, we might think that we want to use codes from the largest class possible (since we might think we’re more likely to get shorter codes).
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).

Theorem 9.5.1
Codeword lengths of any uniquely decodable code (not necessarily instantaneous) must satisfy Kraft inequality:

\[ \sum_{i} D_i \leq 1 \]

Conversely, given a set of codeword lengths that satisfy Kraft, it is possible to construct a uniquely decodable code.
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).
- Could it be true for all uniquely decodable codes?
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).
- Could it be true for all uniquely decodable codes?
- Could larger class of codes have shorter expected codeword lengths?
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).
- Could it be true for all uniquely decodable codes?
- Could larger class of codes have shorter expected codeword lengths?
- Since larger, we might (naïvely) expect that we could do better.
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).
- Could it be true for all uniquely decodable codes?
- Could larger class of codes have shorter expected codeword lengths?
- Since larger, we might (naïvely) expect that we could do better.

**Theorem 9.5.1**

*Codeword lengths of any uniquely decodable code (not. nec. instantaneous) must satisfy Kraft inequality* \( \sum_i D^{-\ell_i} \leq 1 \).
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).
- Could it be true for all uniquely decodable codes?
- Could larger class of codes have shorter expected codeword lengths?
- Since larger, we might (naively) expect that we could do better.

**Theorem 9.5.1**

Codeword lengths of any uniquely decodable code (not nec. instantaneous) must satisfy Kraft inequality \( \sum_i D^{-\ell_i} \leq 1 \). Conversely, given a set of codeword lengths that satisfy Kraft, it is possible to construct a uniquely decodable code.
Kraft revisited

- We proved Kraft inequality is true for instantaneous codes (and vice versa).
- Could it be true for all uniquely decodable codes?
- Could larger class of codes have shorter expected codeword lengths?
- Since larger, we might (naively) expect that we could do better.

**Theorem 9.5.1**

Codeword lengths of *any uniquely decodable code* (not nec. instantaneous) must satisfy Kraft inequality \( \sum_i D^{-\ell_i} \leq 1 \). Conversely, given a set of codeword lengths that satisfy Kraft, it is possible to construct a uniquely decodable code.

**Proof.**

Proof converse we already saw before (given lengths, we can construct a prefix code which is thus uniquely decodable). Thus we only need prove the first part.
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$,
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths \( \ell(x) \), and length of \( k \)-extension \( \ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i) \)
Kraft and uniquely decodable

Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$

we wish to prove that $\sum_{x} D^{-\ell(x)} \leq 1$. 

...
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$ we wish to prove that $\sum_x D^{-\ell(x)} \leq 1$.
- Define $S = \sum_{x \in \mathcal{X}} D^{-\ell(x)}$, $\leq 1$

(9.39)
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$ we wish to prove that $\sum_x D^{-\ell(x)} \leq 1$.
- Define $S = \sum_{x \in \mathcal{X}} D^{-\ell(x)}$, then

\[
(9.39)
\]
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths \( \ell(x) \), and length of \( k \)-extension \( \ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i) \)
we wish to prove that \( \sum_{x} D^{-\ell(x)} \leq 1 \).

- Define \( S = \sum_{x \in X} D^{-\ell(x)} \), then

\[
S^k
\]

(9.39)
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths \( \ell(x) \), and length of \( k \)-extension \( \ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i) \)
  we wish to prove that \( \sum_x D^{-\ell(x)} \leq 1 \).

- Define \( S = \sum_{x \in X} D^{-\ell(x)} \), then

\[
S^k = \left[ \sum_x D^{-\ell(x)} \right]^k
\]

(9.39)
Kraft and uniquely decodable

**Proof of Theorem 9.5.1.**

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$ we wish to prove that $\sum_{x} D^{-\ell(x)} \leq 1$.
- Define $S = \sum_{x \in \mathcal{X}} D^{-\ell(x)}$, then

$$S^k = \left[ \sum_{x} D^{-\ell(x)} \right]^k = \sum_{x_1 \cdots k \in \mathcal{X}^k} D^{-\ell(x_1)} D^{-\ell(x_2)} \cdots D^{-\ell(x_k)} \quad (9.37)$$

$$= \cdots \quad (9.39)$$
Proof of Theorem 9.5.1.

Given: uniquely decodable (not necessarily instantaneous) code with lengths \( \ell(x) \), and length of \( k \)-extension \( \ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i) \) we wish to prove that \( \sum_x D^{-\ell(x)} \leq 1 \).

Define \( S = \sum_{x \in \mathcal{X}} D^{-\ell(x)} \), then

\[
S^k = \left[ \sum_x D^{-\ell(x)} \right]^k = \sum_{x_1:k \in \mathcal{X}^k} D^{-\ell(x_1)} D^{-\ell(x_2)} \ldots D^{-\ell(x_k)} \quad (9.37)
\]

\[
= \sum_{x_1:k \in \mathcal{X}^k} D^{-\left[ \sum_{i=1}^{k} \ell(x_i) \right]} \quad (9.39)
\]
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$ we wish to prove that $\sum_x D^{-\ell(x)} \leq 1$.
- Define $S = \sum_{x \in \mathcal{X}} D^{-\ell(x)}$, then

\[ S^k = \left[ \sum_x D^{-\ell(x)} \right]^k = \sum_{x_1:k \in \mathcal{X}^k} D^{-\ell(x_1)}D^{-\ell(x_2)} \ldots D^{-\ell(x_k)} \tag{9.37} \]

\[ = \sum_{x_1:k \in \mathcal{X}^k} D^{-[\sum_{i=1}^{k} \ell(x_i)]} = \sum_{x_1:k \in \mathcal{X}^k} D^{-\ell(x_1:k)} \tag{9.38} \]

\[ \ldots \tag{9.39} \]
Proof of Theorem 9.5.1.

- Given: uniquely decodable (not necessarily instantaneous) code with lengths $\ell(x)$, and length of $k$-extension $\ell(x, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i)$ we wish to prove that $\sum_x D^{-\ell(x)} \leq 1$.
- Define $S = \sum_{x \in \mathcal{X}} D^{-\ell(x)}$, then

$$S^k = \left[ \sum_x D^{-\ell(x)} \right]^k = \sum_{x_1:k \in \mathcal{X}^k} D^{-\ell(x_1)} D^{-\ell(x_2)} \ldots D^{-\ell(x_k)} \quad (9.37)$$

$$= \sum_{x_1:k \in \mathcal{X}^k} D^{-[\sum_{i=1}^{k} \ell(x_i)]} \sum_{x_1:k \in \mathcal{X}^k} D^{-\ell(x_1:k)} \quad (9.38)$$

$$= \sum_{m=1}^{k\ell_{\max}} a(m) D^{-m} \quad (9.39)$$
Kraft and uniquely decodable

Proof of Theorem 9.5.1.

\[ \sum_{m=1}^{k \ell_{\max}} a(m) D^{-m} \]

(9.39)

where \( \ell_{\max} = \max_x \ell(x) \) is the maximum codeword length.
Kraft and uniquely decodable

Proof of Theorem 9.5.1.

\[
\sum_{m=1}^{k\ell_{\text{max}}} a(m) D^{-m}
\]

(9.39)

- where \( \ell_{\text{max}} = \max_x \ell(x) \) is the maximum codeword length.
- \( a(m) \) = number of source sequences \( x_{1:k} \) mapped into code words of length \( m \), i.e.,

\[
a(m) = \left| \left\{ x_{1:k} \in \mathcal{X}^k : \ell(x_{1:k}) = m \right\} \right|
\]

(9.40)

...
Proof of Theorem 9.5.1.

\[ k \ell_{\max} \sum_{m=1}^{\ell_{\max}} a(m) D^{-m} \]  

(9.39)

- where \( \ell_{\max} = \max_x \ell(x) \) is the maximum codeword length.
- \( a(m) \) = number of source sequences \( x_{1:k} \) mapped into code words of length \( m \), i.e.,

\[ a(m) = \left| \left\{ x_{1:k} \in \mathcal{X}^k : \ell(x_{1:k}) = m \right\} \right| \]  

(9.40)

- There are \( D^m \) codewords of length \( m \), and each of them can have (at most) one associated source sequence (since code is uniquely decodable).
Kraft and uniquely decodable

Proof of Theorem 9.5.1.

\[ k \ell_{\text{max}} \sum_{m=1}^{\ell_{\text{max}}} a(m) D^{-m} \]  \hspace{1cm} (9.39)

- where \( \ell_{\text{max}} = \max_x \ell(x) \) is the maximum codeword length.

- \( a(m) \) = number of source sequences \( x^{1:k} \) mapped into code words of length \( m \), i.e.,

\[ a(m) = \left| \left\{ x^{1:k} \in X^k : \ell(x^{1:k}) = m \right\} \right| \] \hspace{1cm} (9.40)

- There are \( D^m \) codewords of length \( m \), and each of them can have (at most) one associated source sequence (since code is uniquely decodable). Hence, \( a(m) \leq D^m \).
Kraft and uniquely decodable

...proof of Theorem 9.5.1.

- So continuing,

\[ S^k \]

\[ (9.41) \]
...proof of Theorem 9.5.1.

So continuing,

$$S^k = \sum_{m=1}^{k \ell_{\text{max}}} a(m) D^{-m}$$

(9.41)
Kraft and uniquely decodable

... proof of Theorem 9.5.1.

So continuing,

\[ S^k = \sum_{m=1}^{k\ell_{\max}} a(m)D^{-m} \leq \sum_{m=1}^{k\ell_{\max}} D^m D^{-m} \]

(9.41)
proof of Theorem 9.5.1.

So continuing,

\[ S^k = \sum_{m=1}^{k \ell_{\text{max}}} a(m) D^{-m} \leq \sum_{m=1}^{k \ell_{\text{max}}} D^m D^{-m} = k \ell_{\text{max}} \]  

(9.41)

\[ S \geq 1 \Rightarrow S \leq 1 \]
Kraft and uniquely decodable

...proof of Theorem 9.5.1.

So continuing,

\[ S^k = \sum_{m=1}^{k\ell_{\max}} a(m)D^{-m} \leq \sum_{m=1}^{k\ell_{\max}} D^m D^{-m} = k\ell_{\max} \quad \forall k \quad (9.41) \]
proof of Theorem 9.5.1.

So continuing,

\[ S^k = \sum_{m=1}^{k \ell_{\text{max}}} a(m)D^{-m} \leq \sum_{m=1}^{k \ell_{\text{max}}} D^m D^{-m} = k \ell_{\text{max}} \quad \forall k \]  

(9.41)

So, \( S^k \) (exponential in \( k \)) never greater than \( k \ell_{\text{max}} \) (polynomial in \( k \)) \( \Rightarrow S \leq 1 \).
... proof of Theorem 9.5.1.

- So continuing,

\[ S^k = \sum_{m=1}^{k \ell_{\max}} a(m) D^{-m} \leq \sum_{m=1}^{k \ell_{\max}} D^m D^{-m} = k \ell_{\max} \quad \forall k \]  

(9.41)

- So, \( S^k \) (exponential in \( k \)) never greater than \( k \ell_{\max} \) (polynomial in \( k \)) \( \Rightarrow \) \( S \leq 1 \).

- Giving \( S = \sum_{x \in X} D^{-\ell(x)} \leq 1 \).
Summary: uniquely decodable vs. instantaneous codes

- Set of achievable codeword lengths the same for uniquely decodable codes and for instantaneous codes.
Summary: uniquely decodable vs. instantaneous codes

- Set of achievable codeword lengths the same for uniquely decodable codes and for instantaneous codes.
- \( \Rightarrow \) optimal codeword length bound still holds.
Summary: uniquely decodable vs. instantaneous codes

- Set of achievable codeword lengths the same for uniquely decodable codes and for instantaneous codes.
- $\Rightarrow$ optimal codeword length bound still holds.
- In fact, this is not surprising since we can get arbitrarily close to entropy rate already using instantaneous code (e.g., Shannon code) with long block words.
Summary: uniquely decodable vs. instantaneous codes

- Set of achievable codeword lengths the same for uniquely decodable codes and for instantaneous codes.
- $\Rightarrow$ optimal codeword length bound still holds.
- In fact, this is not surprising since we can get arbitrarily close to entropy rate already using instantaneous code (e.g., Shannon code) with long block words.
- So, for distortionless symbol codes, we can then just consider instantaneous codes with impunity.
Summary: uniquely decodable vs. instantaneous codes

- Set of achievable codeword lengths the same for uniquely decodable codes and for instantaneous codes.
- \( \Rightarrow \) optimal codeword length bound still holds.
- In fact, this is not surprising since we can get arbitrarily close to entropy rate already using instantaneous code (e.g., Shannon code) with long block words.
- So, for distortionless symbol codes, we can then just consider instantaneous codes with impunity.
- Soon, we’ll talk about stream codes where we can get the benefit of long block lengths but we don’t have to wait for the end of a block before we start decoding, which is very useful for “streaming” applications like streaming audio/video.
Shannon Code optimal?

- Ex: $\mathcal{X} = \{0, 1\}$ with $p(X = 0) = 10^{-1000} = 1 - p(X = 1)$. 

Shannon lengths:

$$(0) = d \log_2 \frac{1}{10^{-1000}} = 3322 \text{ bits}$$

$$(1) = d \log_2 \frac{1}{1 - 10^{-1000}} = 1 \text{ bit}$$

For symbol 0, we're using 3321 too many bits. In general, for other distributions, one can construct cases where $d \log p_i$ is longer than necessary. Shannon length codes are not optimal integer length prefix codes.
Shannon Code optimal?

- Ex: $\mathcal{X} = \{0, 1\}$ with $p(X = 0) = 10^{-1000} = 1 - p(X = 1)$.
- What are Shannon lengths?

Shannon Code optimal?
Shannon Code optimal?

- **Ex:** $\mathcal{X} = \{0, 1\}$ with $p(X = 0) = 10^{-1000} = 1 - p(X = 1)$.
- **What are Shannon lengths?**
  - $\ell(0) = \lceil \log_2 10^{1000} \rceil = 3322$ bits
Ex: $\mathcal{X} = \{0, 1\}$ with $p(X = 0) = 10^{-1000} = 1 - p(X = 1)$.

What are Shannon lengths?

- $\ell(0) = \lceil \log_2 10^{1000} \rceil = 3322$ bits
- $\ell(1) = \lceil \log_2 (1 - 10^{1000}) \rceil = 1$ bit
Shannon Code optimal?

- Ex: \( \mathcal{X} = \{0, 1\} \) with \( p(X = 0) = 10^{-1000} = 1 - p(X = 1) \).

- What are Shannon lengths?
  - \( \ell(0) = \lceil \log_2 10^{1000} \rceil = 3322 \) bits
  - \( \ell(1) = \lceil \log_2 (1 - 10^{1000}) \rceil = 1 \) bit

- For symbol 0, we’re using 3321 too many bits.
Shannon Code optimal?

- Ex: \( \mathcal{X} = \{0, 1\} \) with \( p(X = 0) = 10^{-1000} = 1 - p(X = 1) \).
- What are Shannon lengths?
  - \( \ell(0) = \lceil \log_2 10^{1000} \rceil = 3322 \) bits
  - \( \ell(1) = \lceil \log_2 (1 - 10^{1000}) \rceil = 1 \) bit
- For symbol 0, we’re using 3321 too many bits.
- In general, for other distributions, one can construct cases where \( \lceil \log_D p_i \rceil \) is longer than necessary.
Shannon Code optimal?

- Ex: $\mathcal{X} = \{0, 1\}$ with $p(X = 0) = 10^{-1000} = 1 - p(X = 1)$.
- What are Shannon lengths?
  - $\ell(0) = \lceil \log_2 10^{1000} \rceil = 3322$ bits
  - $\ell(1) = \lceil \log_2 (1 - 10^{1000}) \rceil = 1$ bit
- For symbol 0, we’re using 3321 too many bits.
- In general, for other distributions, one can construct cases where $\lceil \log_D p_i \rceil$ is longer than necessary.
- Shannon length codes are not optimal integer length prefix codes.
Huffman coding

- A procedure for finding shortest expected length prefix code.
Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
- Here we analyze it armed with the tools of information theory.
Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
- Here we analyze it armed with the tools of information theory.
- Quest: given a $p(x)$, find a code (bit strings and set of lengths) that is as short as possible, and also an instantaneous code (prefix free).
A procedure for finding shortest expected length prefix code.
You’ve probably encountered it in computer science classes (a classic algorithm).
Here we analyze it armed with the tools of information theory.
Quest: given a $p(x)$, find a code (bit strings and set of lengths) that is as short as possible, and also an instantaneous code (prefix free).
We could do this greedily: start at the top and split the potential codewords into even probabilities (i.e., asking the question with highest entropy)
Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
- Here we analyze it armed with the tools of information theory.
- Quest: given a \( p(x) \), find a code (bit strings and set of lengths) that is as short as possible, and also an instantaneous code (prefix free).
- We could do this greedily: start at the top and split the potential codewords into even probabilities (i.e., asking the question with highest entropy)
- This is similar to the game of 20 questions. We have a set of objects, w.l.o.g. the set \( S = \{1, 2, 3, 4, \ldots, m\} \) that occur with frequency proportional to non-negative \( (w_1, w_2, \ldots, w_m) \).
Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
- Here we analyze it armed with the tools of information theory.
- Quest: given a $p(x)$, find a code (bit strings and set of lengths) that is as short as possible, and also an instantaneous code (prefix free).
- We could do this greedily: start at the top and split the potential codewords into even probabilities (i.e., asking the question with highest entropy)
- This is similar to the game of 20 questions. We have a set of objects, w.l.o.g. the set $S = \{1, 2, 3, 4, \ldots, m\}$ that occur with frequency proportional to non-negative $(w_1, w_2, \ldots, w_m)$.
- We wish to determine an object from this class asking as few questions as possible.
Huffman coding

- A procedure for finding shortest expected length prefix code.
- You’ve probably encountered it in computer science classes (a classic algorithm).
- Here we analyze it armed with the tools of information theory.
- Quest: given a $p(x)$, find a code (bit strings and set of lengths) that is as short as possible, and also an instantaneous code (prefix free).
- We could do this greedily: start at the top and split the potential codewords into even probabilities (i.e., asking the question with highest entropy)
- This is similar to the game of 20 questions. We have a set of objects, w.l.o.g. the set $S = \{1, 2, 3, 4, \ldots, m\}$ that occur with frequency proportional to non-negative $(w_1, w_2, \ldots, w_m)$.
- We wish to determine an object from this class asking as few questions as possible.
- Supposing $X \in S$, each question can take the form “Is $X \in A$?” for some $A \subseteq S$. 
Question tree. \( S = \{x_1, x_2, x_3, x_4, x_5\} \).

\[
\begin{align*}
X &\in \{x_2, x_3\} \\
Y &\quad x_2 \quad 0.2 \\
N &\quad x_3 \quad 0.2 \\
X &\in \{x_1\} \\
Y &\quad x_1 \quad 0.3 \\
N &\quad \quad \quad \\
X &\in \{x_4\} \\
Y &\quad x_4 \quad 0.15 \\
N &\quad x_5 \quad 0.15
\end{align*}
\]
20 Questions

- Question tree. $S = \{x_1, x_2, x_3, x_4, x_5\}$.

- How do we construct such a tree?

Charles Sanders Peirce, 1901 said:

Thus twenty skillful hypotheses will ascertain what two hundred thousand stupid ones might fail to do. The secret of the business lies in the caution which breaks a hypothesis up into its smallest logical components, and only risks one of them at a time.
20 Questions

- Question tree. \( S = \{x_1, x_2, x_3, x_4, x_5\} \).

- How do we construct such a tree? Charles Sanders Peirce, 1901 said:
20 Questions

- Question tree. \( S = \{x_1, x_2, x_3, x_4, x_5\} \).

\[
\begin{align*}
X \in \{x_2, x_3\} & \quad X \in \{x_2\} \\
Y & \quad Y \\
& \quad x_2 \quad 0.2 \\
& \quad N \\
& \quad x_3 \quad 0.2 \\
& \quad N \quad x_1 \quad 0.3 \\
& \quad Y \\
& \quad x_4 \quad 0.15 \\
& \quad N \\
& \quad x_5 \quad 0.15
\end{align*}
\]

- How do we construct such a tree? Charles Sanders Peirce, 1901 said:

> Thus twenty skillful hypotheses will ascertain what two hundred thousand stupid ones might fail to do. The secret of the business lies in the caution which breaks a hypothesis up into its smallest logical components, and only risks one of them at a time.
The Greedy Method

- Suggests a greedy method. “Do next whatever currently looks best.”
The Greedy Method

- Suggests a greedy method. “Do next whatever currently looks best.”
- Consider following table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

The question that looks best would infer the most about the distribution, one with the largest entropy. \( H(X | Y_1) = H(X, Y_1) - H(Y_1) \), so choosing a question \( Y_1 \) with large entropy leads to least “residual” uncertainty \( H(X | Y_1) \) about \( X \). Identically, we choose the question \( Y_1 \) with the greatest mutual information about \( X \) since in this case \( I(Y_1; X) = H(X) - H(X | Y_1) = H(Y_1) \). Again, questions take the form “Is \( X_2 \) in \( A \)?” for some \( A \subseteq S \), so choosing a yes/no (binary) question means choosing the set \( A \).
The Greedy Method

- Suggests a greedy method. “Do next whatever currently looks best.”
- Consider following table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

- The question that looks best would infer the most about the distribution, one with the largest entropy.
The Greedy Method

- Suggests a greedy method. “Do next whatever currently looks best.”
- Consider following table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

- The question that looks best would infer the most about the distribution, one with the largest entropy.
- $H(X|Y_1) = H(X, Y_1) - H(Y_1) = H(X) - H(Y_1)$, so choosing a question $Y_1$ with large entropy leads to least “residual” uncertainty $H(X|Y_1)$ about $X$. 

The Greedy Method

- Suggests a greedy method. “Do next whatever currently looks best.”
- Consider following table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

- The question that looks best would infer the most about the distribution, one with the largest entropy.

\[
H(X|Y_1) = H(X, Y_1) - H(Y_1) = H(X) - H(Y_1),
\]

so choosing a question \( Y_1 \) with large entropy leads to least “residual” uncertainty \( H(X|Y_1) \) about \( X \).

- Identically, we choose the question \( Y_1 \) with the greatest mutual information about \( X \) since in this case

\[
I(Y_1; X) = H(X) - H(X|Y_1) = H(Y_1).
\]

\[
= H(Y_1) - H(Y_1|X)
\]
The Greedy Method

- Suggests a greedy method. “Do next whatever currently looks best.”
- Consider following table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

- The question that looks best would infer the most about the distribution, one with the largest entropy.

$$H(X|Y_1) = H(X, Y_1) - H(Y_1) = H(X) - H(Y_1),$$

so choosing a question $Y_1$ with large entropy leads to least “residual” uncertainty $H(X|Y_1)$ about $X$.

- Identically, we choose the question $Y_1$ with the greatest mutual information about $X$ since in this case

$$I(Y_1; X) = H(X) - H(X|Y_1) = H(Y_1).$$

- Again, questions take the form “Is $X \in A$?” for some $A \subseteq S$, so choosing a yes/no (binary) question means choosing the set $A$. 
The Greedy Method

- We’ll use greedy, and choose the question (set) with the greatest entropy.
The Greedy Method

- We’ll use greedy, and choose the question (set) with the greatest entropy.

- If we consider the partition \( \{a, b, c, d, e, f, g\} = \{a, b, c, d\} \cup \{e, f, g\} \), the question “Is \( X \in \{e, f, g\}\)?” would have maximum entropy since \( p(X \in \{a, b, c, d\}) = p(X \in \{e, f, g\}) = 0.5 \).
The Greedy Method

- We’ll use greedy, and choose the question (set) with the greatest entropy.

- If we consider the partition
  \[ \{a, b, c, d, e, f, g\} = \{a, b, c, d\} \cup \{e, f, g\} \], the question “Is \(X \in \{e, f, g\}\)” would have maximum entropy since
  \[ p(X \in \{a, b, c, d\}) = p(X \in \{e, f, g\}) = 0.5. \]

- This question corresponds to random variable \(Y_1 = 1_{\{X \in \{e, f, g\}\}}\) so \(H(Y_1) = 1\) and this would be considered a good question (as good as it gets for binary r.v.).
The Greedy Method

- We’ll use greedy, and choose the question (set) with the greatest entropy.
- If we consider the partition 
  \( \{a, b, c, d, e, f, g\} = \{a, b, c, d\} \cup \{e, f, g\} \), the question “Is \( X \in \{e, f, g\}\)?” would have maximum entropy since 
  \( p(X \in \{a, b, c, d\}) = p(X \in \{e, f, g\}) = 0.5 \).
- This question corresponds to random variable \( Y_1 = 1_{\{X \in \{e,f,g\}\}} \) so 
  \( H(Y_1) = 1 \) and this would be considered a good question (as good as it gets for binary r.v.).
- The next question depends on the outcome of the first, and we have either 
  \( Y_1 = 0 (\equiv X \in \{a, b, c, d\}) \) or \( Y_1 = 1 (\equiv X \in \{e, f, g\}) \).
The Greedy Tree

- If \( Y_1 = 0 \) then we can split to maximize entropy as follows: partition \( \{a, b, c, d\} = \{a, b\} \cup \{c, d\} \) since \( p(\{a, b\}) = p(\{c, d\}) = 1/4 \).
The Greedy Tree

- If $Y_1 = 0$ then we can split to maximize entropy as follows: partition 
  $\{a, b, c, d\} = \{a, b\} \cup \{c, d\}$ since $p(\{a, b\}) = p(\{c, d\}) = \frac{1}{4}$.

- This question corresponds to random variable $Y_2 = 1_{\{X \in \{c, d\}}$ so 
  $H(Y_2 | Y_1 = 0) = 1$ and this would also be considered a good
  question (as good as it gets).
The Greedy Tree

- If $Y_1 = 0$ then we can split to maximize entropy as follows: partition
  \( \{a, b, c, d\} = \{a, b\} \cup \{c, d\} \) since \( p(\{a, b\}) = p(\{c, d\}) = 1/4 \).

- This question corresponds to random variable \( Y_2 = 1_{\{X \in \{c, d\}\}} \) so
  \( H(Y_2|Y_1 = 0) = 1 \) and this would also be considered a good question (as good as it gets).

- If $Y_1 = 1$, then we need to partition the set \( \{e, f, g\} \).
The Greedy Tree

- If $Y_1 = 0$ then we can split to maximize entropy as follows: partition \( \{a, b, c, d\} = \{a, b\} \cup \{c, d\} \) since $p(\{a, b\}) = p(\{c, d\}) = 1/4$.
- This question corresponds to random variable $Y_2 = 1_{\{X \in \{c,d\}}$ so $H(Y_2|Y_1 = 0) = 1$ and this would also be considered a good question (as good as it gets).
- If $Y_1 = 1$, then we need to partition the set $\{e, f, g\}$.
- We can do this in one of three ways:

<table>
<thead>
<tr>
<th>case</th>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>split</td>
<td>${e}, {f, g}$</td>
<td>${e, f}, {g}$</td>
<td>${e, g}, {f}$</td>
</tr>
<tr>
<td>prob</td>
<td>$(0.47, 0.03)$</td>
<td>$(0.48, 0.2)$</td>
<td>$(0.49, 0.1)$</td>
</tr>
<tr>
<td>$H(Y_2</td>
<td>Y_1 = 1)$</td>
<td>0.3274</td>
<td>0.2423</td>
</tr>
</tbody>
</table>
The Greedy Tree

- If $Y_1 = 0$ then we can split to maximize entropy as follows: partition 
  $\{a, b, c, d\} = \{a, b\} \cup \{c, d\}$ since $p(\{a, b\}) = p(\{c, d\}) = 1/4$.
- This question corresponds to random variable $Y_2 = 1_{\{X \in \{c,d\}}$ so 
  $H(Y_2|Y_1 = 0) = 1$ and this would also be considered a good question (as good as it gets).
- If $Y_1 = 1$, then we need to partition the set $\{e, f, g\}$.
- We can do this in one of three ways:

<table>
<thead>
<tr>
<th>case</th>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>split</td>
<td>${e}, {f, g}$</td>
<td>${e, f}, {g}$</td>
<td>${e, g}, {f}$</td>
</tr>
<tr>
<td>prob</td>
<td>(0.47,0.03)</td>
<td>(0.48,0.2)</td>
<td>(0.49,0.1)</td>
</tr>
<tr>
<td>$H(Y_2</td>
<td>Y_1 = 1)$</td>
<td>0.3274</td>
<td>0.2423</td>
</tr>
</tbody>
</table>

- Thus, we would choose case I for $Y_2$ since that is the maximum entropy question. Thus, we get $H(Y_2|Y_1 = 1) = 0.3274$
The Greedy Tree

- If $Y_1 = 0$ then we can split to maximize entropy as follows: partition 
  $\{a, b, c, d\} = \{a, b\} \cup \{c, d\}$ since $p(\{a, b\}) = p(\{c, d\}) = 1/4$.

- This question corresponds to random variable $Y_2 = 1_{\{X \in \{c,d\}}$ so $H(Y_2|Y_1 = 0) = 1$ and this would also be considered a good question (as good as it gets).

- If $Y_1 = 1$, then we need to partition the set $\{e, f, g\}$.

- We can do this in one of three ways:

| case | I                  | II                  | III
|------|--------------------|---------------------|------
| split | $\{e\}, \{f, g\}$ | $\{e, f\}, \{g\}$ | $\{e, g\}, \{f\}$
| prob  | (0.47, 0.03)       | (0.48, 0.2)         | (0.49, 0.1) 
| $H(Y_2|Y_1 = 1)$ | 0.3274             | 0.2423              | 0.1414

- Thus, we would choose case I for $Y_2$ since that is the maximum entropy question. Thus, we get $H(Y_2|Y_1 = 1) = 0.3274$.

- Also, $H(X|Y_2, Y_1) = H(X, Y_2|Y_1) - H(Y_2|Y_1) = H(X|Y_1) - H(Y_2|Y_1) = H(X) - H(Y_2|Y_1) - H(Y_1)$.
The Greedy Tree

- Once we get to sets of size 2, we only have one possible question. Greedy strategy always greedily chooses what currently looks best, ignoring future. Latter questions must live with what is available.

<table>
<thead>
<tr>
<th>set</th>
<th>split</th>
<th>probabilities</th>
<th>conditional entropies</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b, c, d, e, f}</td>
<td>{}</td>
<td>(0.5,0.5)</td>
<td>$H(Y_1) = 1$</td>
</tr>
<tr>
<td>{a, b, c, d}</td>
<td>{}, {e, f, g}</td>
<td>(0.25,0.25)</td>
<td>$H(Y_2</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{}, {c, d}</td>
<td>(0.01,0.24)</td>
<td>$H(Y_3</td>
</tr>
<tr>
<td>{c, d}</td>
<td>{}, {e}</td>
<td>(0.05,0.20)</td>
<td>$H(Y_3</td>
</tr>
<tr>
<td>{e}</td>
<td>{}</td>
<td>(0.47)</td>
<td>$H(Y_3</td>
</tr>
<tr>
<td>{f, g}</td>
<td>{}, {}</td>
<td>(0.01,0.02)</td>
<td>$H(Y_3</td>
</tr>
</tbody>
</table>

Also note, $H(X) = H(Y_1, Y_2, Y_3) = 1.9323$ and recall $H(Y_1, Y_2, Y_3) = H(Y_1) + H(Y_2|Y_1) + H(Y_3|Y_1, Y_2)$ (9.42) + $\sum_{i, j} H(Y_3|Y_1=i, Y_2=j) p(Y_1=i, Y_2=j)$ (9.43).
The Greedy Tree

- Once we get to sets of size 2, we only have one possible question. Greedy strategy always greedily chooses what currently looks best, ignoring future. Latter questions must live with what is available.
- Summarizing all questions/splits, & their conditional entropies:

<table>
<thead>
<tr>
<th>set</th>
<th>split</th>
<th>prob.</th>
<th>( H(Y_1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b, c, d, e, f}</td>
<td></td>
<td>(0.5,0.5)</td>
<td></td>
</tr>
<tr>
<td>{a, b, c, d}</td>
<td>{e, f, g}</td>
<td>(0.25,0.25)</td>
<td></td>
</tr>
<tr>
<td>{a, b}</td>
<td>{c, d}</td>
<td>(0.01,0.24)</td>
<td></td>
</tr>
<tr>
<td>{e, f, g}</td>
<td></td>
<td>(0.47,0.3)</td>
<td>( 0 )</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a}</td>
<td>(0.01,0.24)</td>
<td>( 0 )</td>
</tr>
<tr>
<td>{c, d}</td>
<td>{c}</td>
<td>(0.05,0.20)</td>
<td>( 0 )</td>
</tr>
<tr>
<td>{e}</td>
<td></td>
<td>(0.47)</td>
<td>( 0 )</td>
</tr>
<tr>
<td>{f, g}</td>
<td></td>
<td>(0.01,0.02)</td>
<td>( 0 )</td>
</tr>
</tbody>
</table>
The Greedy Tree

- Once we get to sets of size 2, we only have one possible question. Greedy strategy always greedily chooses what currently looks best, ignoring future. Latter questions must live with what is available.
- Summarizing all questions/splits, & their conditional entropies:

<table>
<thead>
<tr>
<th>set</th>
<th>split</th>
<th>probabilities</th>
<th>conditional entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b, c, d, e, f}</td>
<td>{a, b, c, d}</td>
<td>(0.5, 0.5)</td>
<td>H(Y_1) = 1</td>
</tr>
<tr>
<td></td>
<td>{e, f, g}</td>
<td>(0.47, 0.3)</td>
<td>H(Y_2</td>
</tr>
<tr>
<td></td>
<td>{a, b}</td>
<td>(0.01, 0.24)</td>
<td>H(Y_3</td>
</tr>
<tr>
<td></td>
<td>{c, d}</td>
<td>(0.05, 0.20)</td>
<td>H(Y_3</td>
</tr>
<tr>
<td></td>
<td>{e}</td>
<td>(0.47)</td>
<td>H(Y_3</td>
</tr>
<tr>
<td></td>
<td>{f, g}</td>
<td>(0.01, 0.02)</td>
<td>H(Y_3</td>
</tr>
</tbody>
</table>

Also note, \( H(X) = H(Y_1, Y_2, Y_3) = 1 \).

and recall
\[
H(Y_1, Y_2, Y_3) = H(Y_1) + H(Y_2 | Y_1) + H(Y_3 | Y_1, Y_2) \]
\[
+ \sum_{i,j} H(Y_3 | Y_1 = i, Y_2 = j) p(Y_1 = i, Y_2 = j) \]

Prof. Jeff Bilmes
The Greedy Tree

- Once we get to sets of size 2, we only have one possible question. Greedy strategy always greedily chooses what currently looks best, ignoring future. Latter questions must live with what is available.

- Summarizing all questions/splits, & their conditional entropies:

<table>
<thead>
<tr>
<th>set</th>
<th>split</th>
<th>probabilities</th>
<th>conditional entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b, c, d, e, f}</td>
<td>{a, b, c, d}, {e, f, g}</td>
<td>(0.5,0.5)</td>
<td>(H(Y_1) = 1)</td>
</tr>
<tr>
<td>{a, b, c, d}</td>
<td>{a, b}, {c, d}</td>
<td>(0.25,0.25)</td>
<td>(H(Y_2</td>
</tr>
<tr>
<td>{e, f, g}</td>
<td>{e}, {f, g}</td>
<td>(0.47,0.3)</td>
<td>(H(Y_2</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a}, {b}</td>
<td>(0.01,0.24)</td>
<td>(H(Y_3</td>
</tr>
<tr>
<td>{c, d}</td>
<td>{c}, {d}</td>
<td>(0.05,0.20)</td>
<td>(H(Y_3</td>
</tr>
<tr>
<td>{e}</td>
<td>{e}</td>
<td>(0.47)</td>
<td>(H(Y_3</td>
</tr>
<tr>
<td>{f, g}</td>
<td>{f}, {g}</td>
<td>(0.01,0.02)</td>
<td>(H(Y_3</td>
</tr>
</tbody>
</table>

Also note,

\[
H(X) = H(Y_1, Y_2, Y_3) = 1.9323
\]

and recall

\[
H(Y_1, Y_2, Y_3) = H(Y_1) + \sum_{i=0}^{1} H(Y_2|Y_1=i) \cdot p(Y_1=i) + \sum_{i,j=0}^{1} H(Y_3|Y_1=i, Y_2=j) \cdot p(Y_1=i, Y_2=j)
\]
The Greedy Tree

- Once we get to sets of size 2, we only have one possible question. Greedy strategy always greedily chooses what currently looks best, ignoring future. Latter questions must live with what is available.

- Summarizing all questions/splits, & their conditional entropies:

<table>
<thead>
<tr>
<th>set</th>
<th>split</th>
<th>probabilities</th>
<th>conditional entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b, c, d, e, f}</td>
<td>{a, b, c, d}, {e, f, g}</td>
<td>(0.5,0.5)</td>
<td>(H(Y_1) = 1)</td>
</tr>
<tr>
<td>{a, b, c, d}</td>
<td>{a, b}, {c, d}</td>
<td>(0.25,0.25)</td>
<td>(H(Y_2</td>
</tr>
<tr>
<td>{e, f, g}</td>
<td>{e}, {f, g}</td>
<td>(0.47,0.3)</td>
<td>(H(Y_2</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a}, {b}</td>
<td>(0.01,0.24)</td>
<td>(H(Y_3</td>
</tr>
<tr>
<td>{c, d}</td>
<td>{c}, {d}</td>
<td>(0.05,0.20)</td>
<td>(H(Y_3</td>
</tr>
<tr>
<td>{e}</td>
<td>{e}</td>
<td>(0.47)</td>
<td>(H(Y_3</td>
</tr>
<tr>
<td>{f, g}</td>
<td>{f}, {g}</td>
<td>(0.01,0.02)</td>
<td>(H(Y_3</td>
</tr>
</tbody>
</table>

- Also note, \(H(X) = H(Y_1, Y_2, Y_3) = 1.9323\) and recall
The Greedy Tree

- Once we get to sets of size 2, we only have one possible question. Greedy strategy always greedily chooses what currently looks best, ignoring future. Latter questions must live with what is available.

- Summarizing all questions/splits, & their conditional entropies:

- Also note, $H(X) = H(Y_1, Y_2, Y_3) = 1.9323$ and recall

\[
H(Y_1, Y_2, Y_3) = H(Y_1) + H(Y_2|Y_1) + H(Y_3|Y_1, Y_2)
\]

\[
= H(Y_1) + \sum_{i \in \{0,1\}} H(Y_2|Y_1 = i)p(Y_1 = i)
\]

\[
+ \sum_{i,j \in \{0,1\}} H(Y_3|Y_1 = i, Y_2 = j)p(Y_1 = i, Y_2 = j)
\]
The Greedy Tree

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

This leads to the following (top-down greedily constructed) tree:
The Greedy Tree

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

This leads to the following (top-down greedily constructed) tree:

```
{a, b, c, d, e, f, g}
```

The expected length of this code $E = 2.5300$.

Entropy: $H = 1.9323$.

Code efficiency $H/E = 1.9323 / 2.5300 = 0.7638$.

Can we do better?
The Greedy Tree

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

This leads to the following (top-down greedily constructed) tree:

\[ \{a, b, c, d, e, f, g\} \]

The expected length of this code \( E\ell = 2.5300 \)
The Greedy Tree

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

This leads to the following (top-down greedily constructed) tree:

\{a, b, c, d, e, f, g\}

The expected length of this code \(E\ell = 2.5300\).

Entropy: \(H = 1.9323\).
The Greedy Tree

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

This leads to the following (top-down greedily constructed) tree:

\[ \{a, b, c, d, e, f, g\} \]

The expected length of this code \( E\ell = 2.5300 \).

Entropy: \( H = 1.9323 \).

Code efficiency \( H/E\ell = 1.9323/2.5300 = 0.7638 \).
The Greedy Tree

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0.01</td>
<td>0.24</td>
<td>0.05</td>
<td>0.20</td>
<td>0.47</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

- This leads to the following (top-down greedily constructed) tree:

{a, b, c, d, e, f, g}

- The expected length of this code $E\ell = 2.5300$.
- Entropy: $H = 1.9323$.
- Code efficiency $H/E\ell = 1.9323/2.5300 = 0.7638$.
- Can we do better?
The Greedy Tree vs. Huffman Tree

- Left is greedy, right is Huffman
The Greedy Tree vs. Huffman Tree

- Left is greedy, right is Huffman

{a, b, c, d, e, f, g}

The Hu man lengths have $E_{\text{hu}} = 1.9700$. Efficiency of Hu man code: $H/E_{\text{hu}} = 1.9323/1.9700 = 0.9809$. Key problem: Greedy procedure is not optimal in this case.
The Greedy Tree vs. Huffman Tree

- Left is greedy, right is Huffman

The Huffman lengths have $E\ell_{huffman} = 1.9700$
The Greedy Tree vs. Huffman Tree

- Left is greedy, right is Huffman

The Huffman lengths have $E \ell_{huffman} = 1.9700$

Efficiency of Huffman code: $H/E \ell_{huffman} = 1.9323/1.9700 = 0.9809$
The Greedy Tree vs. Huffman Tree

- Left is greedy, right is Huffman

The Huffman lengths have $E\ell_{\text{huffman}} = 1.9700$

Efficiency of Huffman code: $H/E\ell_{\text{huffman}} = 1.9323/1.9700 = 0.9809$

Key problem: Greedy procedure is not optimal in this case.
Why does starting from the top and splitting as such non-optimal? Where can it go wrong?
Why does starting from the top and splitting as such non-optimal? Where can it go wrong?

Ex: There may be many ways to get a $\approx 50\%$ split (to achieve high entropy) once done, the split is irrevocable and there is no way to know if the consequences of that split might hurt down the line.
The Huffman code tree procedure
The Huffman code tree procedure

1. take the two least probable symbols in the alphabet.
The Huffman code tree procedure

1. take the two least probable symbols in the alphabet.
2. These two will be given the longest codewords, will have equal length, and will differ in the last digit.
The Huffman code tree procedure

1. take the two least probable symbols in the alphabet.
2. These two will be given the longest codewords, will have equal length, and will differ in the last digit.
3. Combine these two symbols into a joint symbol having probability equal to the sum, add the joint symbol and then remove the two symbols, and repeat.
The Huffman code tree procedure

1. Take the two least probable symbols in the alphabet.
2. These two will be given the longest codewords, will have equal length, and will differ in the last digit.
3. Combine these two symbols into a joint symbol having probability equal to the sum, add the joint symbol and then remove the two symbols, and repeat.

Note that it is bottom up (agglomerative clustering) rather than top down (greedy splitting).
Huffman

- Ex: \( \mathcal{X} = \{1, 2, 3, 4, 5\} \) with probabilities \( \{1/4, 1/4, 1/5, 3/20, 3/20\} \).
Huffman

- Ex: \( \mathcal{X} = \{1, 2, 3, 4, 5\} \) with probabilities \( \{1/4, 1/4, 1/5, 3/20, 3/20\} \).
- So 4 and 5 should have longest code length
Huffman

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.
- So 4 and 5 should have longest code length
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\mathcal{X}$</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
</tr>
</tbody>
</table>
Huffman

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.
- So 4 and 5 should have longest code length
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\mathcal{X}$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.

So 4 and 5 should have longest code length

We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\mathcal{X}$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>---</td>
<td>0.25</td>
<td>---</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>---</td>
<td>0.25</td>
<td>0</td>
<td>0.45</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>---</td>
<td>0.2</td>
<td>1</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
<td>0</td>
<td>0.3</td>
<td>---</td>
<td>0.3</td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Huffman

- Ex: \( \mathcal{X} = \{1, 2, 3, 4, 5\} \) with probabilities \( \{1/4, 1/4, 1/5, 3/20, 3/20\} \).
- So 4 and 5 should have longest code length
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>( \mathcal{X} )</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
<th>step 3</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.55</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.25</td>
<td>0.45</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
<td>0.3</td>
<td>0</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>
Huffman

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{5}, \frac{3}{20}, \frac{3}{20}\}$.
- So 4 and 5 should have longest code length.
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\mathcal{X}$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
<th>step 3</th>
<th>prob</th>
<th>step 4</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.55</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.45</td>
<td>0.45</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>0.45</td>
<td>0.45</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

So we have $E = 2.3$ bits and $H = 2.2855$ bits, as you can see this code does pretty well (close to entropy).

Some code lengths are shorter/longer than $I(x) = \log \frac{1}{p(x)}$. Construction is similar for $D > 2$, in such case we might use dummy symbols in alphabet.
Huffman

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.
- So 4 and 5 should have longest code length
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\log \frac{1}{p(x)}$</th>
<th>length</th>
<th>codeword</th>
<th>$\mathcal{X}$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
<th>step 3</th>
<th>prob</th>
<th>step 4</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2</td>
<td>00</td>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.55</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.45</td>
<td>0.45</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>0.2</td>
<td>0.2</td>
<td></td>
<td>0.45</td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>010</td>
<td>4</td>
<td>0.15</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>011</td>
<td>5</td>
<td>0.15</td>
<td>0</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- So we have $E \ell = 2.3$ bits and $H = 2.2855$ bits, as you can see this code does pretty well (close to entropy).
Huffman

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.
- So 4 and 5 should have longest code length.
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\log \frac{1}{p(x)}$</th>
<th>length</th>
<th>codeword</th>
<th>$\mathcal{X}$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
<th>step 3</th>
<th>prob</th>
<th>step 4</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2</td>
<td>00</td>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.55</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0.45</td>
<td>0.45</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
<td>0.45</td>
<td>0.45</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>010</td>
<td>4</td>
<td>0.15</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>011</td>
<td>5</td>
<td>0.15</td>
<td>1</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- So we have $E\ell = 2.3$ bits and $H = 2.2855$ bits, as you can see this code does pretty well (close to entropy).
- Some code lengths are shorter/longer than $I(x) = \log \frac{1}{p(x)}$. 
Huffman

- Ex: $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $\{1/4, 1/4, 1/5, 3/20, 3/20\}$.
- So 4 and 5 should have longest code length.
- We build the tree from left to right.

<table>
<thead>
<tr>
<th>$\log \frac{1}{p(x)}$</th>
<th>length</th>
<th>codeword</th>
<th>$\mathcal{X}$</th>
<th>prob</th>
<th>step 1</th>
<th>prob</th>
<th>step 2</th>
<th>prob</th>
<th>step 3</th>
<th>prob</th>
<th>step 4</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2</td>
<td>00</td>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.45</td>
<td>0.55</td>
<td>0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0.45</td>
<td>0.45</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>010</td>
<td>4</td>
<td>0.15</td>
<td>0</td>
<td>0.45</td>
<td>0.3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>3</td>
<td>011</td>
<td>5</td>
<td>0.15</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- So we have $E\ell = 2.3$ bits and $H = 2.2855$ bits, as you can see this code does pretty well (close to entropy).
- Some code lengths are shorter/longer than $I(x) = \log \frac{1}{p(x)}$.
- Construction is similar for $D > 2$, in such case we might use dummy symbols in alphabet.
More Huffman vs. Shannon

- Shannon code lengths $\ell_i = \lceil \log \frac{1}{p_i} \rceil$ we saw are not optimal – more realistic example, binary alphabet with probabilities $p(a) = 0.9999$ and $p(b) = 1 - 0.9999$ lead to lengths $\ell_a = 1$ and $\ell_b = 14$ bits, with $E\ell = 1.0013 > 1$. 
More Huffman vs. Shannon

- Shannon code lengths $\ell_i = \lceil \log 1/p_i \rceil$ we saw are not optimal – more realistic example, binary alphabet with probabilities $p(a) = 0.9999$ and $p(b) = 1 - 0.9999$ lead to lengths $\ell_a = 1$ and $\ell_b = 14$ bits, with $E\ell = 1.0013 > 1$.

- Optimal code lengths are not always $\leq \lceil \log 1/p_i \rceil$. 
Shannon code lengths $\ell_i = \lceil \log 1/p_i \rceil$ we saw are not optimal – more realistic example, binary alphabet with probabilities $p(a) = 0.9999$ and $p(b) = 1 - 0.9999$ lead to lengths $\ell_a = 1$ and $\ell_b = 14$ bits, with $E\ell = 1.0013 > 1$.

Optimal code lengths are not always $\leq \lceil \log 1/p_i \rceil$.

Consider $X$ with probabilities $(1/3, 1/3, 1/4, 1/12)$ with $H = 1.8554$. 
Shannon code lengths $\ell_i = \lceil \log 1/p_i \rceil$ we saw are not optimal – more realistic example, binary alphabet with probabilities $p(a) = 0.9999$ and $p(b) = 1 - 0.9999$ lead to lengths $\ell_a = 1$ and $\ell_b = 14$ bits, with $E\ell = 1.0013 > 1$.

Optimal code lengths are not always $\leq \lceil \log 1/p_i \rceil$.

Consider $X$ with probabilities $(1/3, 1/3, 1/4, 1/12)$ with $H = 1.8554$.

Huffman lengths are either $L_{h1} = (2, 2, 2, 2)$ or $L_{h2} = (1, 2, 3, 3)$
More Huffman vs. Shannon

- Shannon code lengths $\ell_i = \lceil \log 1/p_i \rceil$ we saw are not optimal — more realistic example, binary alphabet with probabilities $p(a) = 0.9999$ and $p(b) = 1 - 0.9999$ lead to lengths $\ell_a = 1$ and $\ell_b = 14$ bits, with $E\ell = 1.0013 > 1$.

- Optimal code lengths are not always $\leq \lceil \log 1/p_i \rceil$.

- Consider $X$ with probabilities $(1/3, 1/3, 1/4, 1/12)$ with $H = 1.8554$.

- Huffman lengths are either $L_{h1} = (2, 2, 2, 2)$ or $L_{h2} = (1, 2, 3, 3)$

- But $\lceil \log 1/p_3 \rceil = \lceil -\log(1/4) \rceil = 2 < 3$. Shannon lengths are $L_s = (2, 2, 2, 4)$ with $EL_s = 2.1557 > 2$. 
More Huffman vs. Shannon

- Shannon code lengths $\ell_i = \lceil \log 1/p_i \rceil$ we saw are not optimal – more realistic example, binary alphabet with probabilities $p(a) = 0.9999$ and $p(b) = 1 - 0.9999$ lead to lengths $\ell_a = 1$ and $\ell_b = 14$ bits, with $E\ell = 1.0013 > 1$.

- Optimal code lengths are not always $\leq \lceil \log 1/p_i \rceil$.

- Consider $X$ with probabilities $(1/3, 1/3, 1/4, 1/12)$ with $H = 1.8554$.

- Huffman lengths are either $L_{h1} = (2, 2, 2, 2)$ or $L_{h2} = (1, 2, 3, 3)$

- But $\lceil \log 1/p_3 \rceil = \lceil - \log(1/4) \rceil = 2 < 3$. Shannon lengths are $L_s = (2, 2, 2, 4)$ with $EL_s = 2.1557 > 2$.

- In general, a particular codeword for the optimal code might be longer than Shannon’s length, but of course this is not true on average.
Huffman is optimal, i.e., $\sum_i p_i l_i$ is minimal, for integer lengths.
Optimality of Huffman

- Huffman is optimal, i.e., $\sum_i p_i \ell_i$ is minimal, for integer lengths.
- To show this:
Optimality of Huffman

- Huffman is optimal, i.e., $\sum_i p_i \ell_i$ is minimal, for integer lengths.
- To show this:
  - First show lemma that some optimal codes have certain properties (not all, but that $\exists$ optimal code with these properties).
Optimality of Huffman

- Huffman is optimal, i.e., $\sum_i p_i \ell_i$ is minimal, for integer lengths.
- To show this:
  1. First show lemma that some optimal codes have certain properties (not all, but that $\exists$ optimal code with these properties).
  2. Given a code $C_m$ for $m$ symbols, that has said properties, produce new simpler code satisfying lemma and is simpler to optimize.
Huffman is optimal, i.e., \( \sum_i p_i \ell_i \) is minimal, for integer lengths.

To show this:

1. First show lemma that some optimal codes have certain properties (not all, but that \( \exists \) optimal code with these properties).
2. Given a code \( C_m \) for \( m \) symbols, that has said properties, produce new simpler code satisfying lemma and is simpler to optimize.
3. Ultimately get down to simple case of two symbols which are obvious to optimize.
**Optimality of Huffman**

**Lemma 9.6.1**

For all distributions, \( \exists \) an optimal instantaneous code (i.e., minimal expected length) simultaneously satisfying:

1. if \( p_j > p_k \) then \( l_j \leq l_k \) (i.e., the more probable symbol does not have a longer length)
2. The two longest codewords have the same length
3. Two longest codewords differ only in last bit and correspond to the two least likely symbols.

**Proof.**

- Suppose \( C_m \) is optimal code (so \( L(C_m) \) is minimum) and choose \( j, k \) such that \( p_j > p_k \). Need to show \( \exists \) code with \( l_j \leq l_k \).
- Consider \( C'_m \) with codewords \( j \) and \( k \) swapped meaning
  \[
  l'_j = l_k \quad \text{and} \quad l'_k = l_j \tag{9.44}
  \]
  which can only make the code longer, so \( L(C'_m) \geq L(C_m) \) …
Proof of lemma 9.6.1.

With this swap, since $L(C_m)$ is minimal, we have

$$0$$

(9.49)
With this swap, since \( L(C_m) \) is minimal, we have

\[
0 \leq L(C'_m) - L(C_m)
\]

(9.49)
Optimality of Huffman

... proof of lemma 9.6.1.

- With this swap, since $L(C_m')$ is minimal, we have

$$0 \leq L(C_m') - L(C_m) = \sum_i p_i l_i' - \sum_i p_i l_i$$ (9.45)

(9.49)
Optimality of Huffman

... proof of lemma 9.6.1.

- With this swap, since $L(C_m')$ is minimal, we have

$$0 \leq L(C_m') - L(C_m) = \sum_i p_i \ell_i' - \sum_i p_i \ell_i$$

(9.45)

$$= p_j \ell_j' + p_k \ell_k' - p_j \ell_j - p_k \ell_k$$

(9.46)

(9.49)

...
With this swap, since $L(C_m')$ is minimal, we have

$$0 \leq L(C_m') - L(C_m) = \sum_i p_i \ell'_i - \sum_i p_i \ell_i$$

(9.45)

$$= p_j \ell'_j + p_k \ell'_k - p_j \ell_j - p_k \ell_k$$

(9.46)

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k$$

(9.47)

(9.49)
With this swap, since $L(C'_m)$ is minimal, we have

\[0 \leq L(C'_m) - L(C_m) = \sum_i p_i \ell'_i - \sum_i p_i \ell_i \]  \hspace{1cm} (9.45)

\[= p_j \ell'_j + p_k \ell'_k - p_j \ell_j - p_k \ell_k \]  \hspace{1cm} (9.46)

\[= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k \]  \hspace{1cm} (9.47)

\[= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j) \]  \hspace{1cm} (9.48)

\[= (p_j - p_k) (\ell_k - \ell_j) \]  \hspace{1cm} (9.49)
With this swap, since $L(C_m')$ is minimal, we have

$$0 \leq L(C_m') - L(C_m) = \sum_i p_i \ell_i' - \sum_i p_i \ell_i$$

(9.45)

$$= p_j \ell_j' + p_k \ell_k' - p_j \ell_j - p_k \ell_k$$

(9.46)

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k$$

(9.47)

$$= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j)$$

(9.48)

$$= (p_j - p_k) (\ell_k - \ell_j)$$

(9.49)
... proof of lemma 9.6.1.

- With this swap, since $L(C_m)$ is minimal, we have

$$0 \leq L(C'_m) - L(C_m) = \sum_i p_i \ell'_i - \sum_i p_i \ell_i$$  \hspace{1cm} (9.45)

$$= p_j \ell'_j + p_k \ell'_k - p_j \ell_j - p_k \ell_k$$  \hspace{1cm} (9.46)

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k$$  \hspace{1cm} (9.47)

$$= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j)$$  \hspace{1cm} (9.48)

$$= (p_j - p_k) (\ell_k - \ell_j) \geq 0$$  \hspace{1cm} (9.49)

Thus, $\ell'_k \leq \ell'_j$ when $p_j > p_k$ and the code satisfies property 1. In fact, this property is true for all optimal codes (stronger than the "there exists" statement of the theorem).
... proof of lemma 9.6.1.

With this swap, since $L(C_m')$ is minimal, we have:

$$0 \leq L(C_m') - L(C_m) = \sum_i p_i \ell_i' - \sum_i p_i \ell_i$$  \hspace{1cm} (9.45)

$$= p_j \ell_j' + p_k \ell_k' - p_j \ell_j - p_k \ell_k$$  \hspace{1cm} (9.46)

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k$$  \hspace{1cm} (9.47)

$$= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j)$$  \hspace{1cm} (9.48)

$$= (p_j - p_k) (\ell_k - \ell_j) \geq 0$$

$$\geq 0$$

Thus, $\ell_j'$ when $p_j > p_k$ and the code satisfies property 1. In fact, this property is true for all optimal codes (stronger than the “there exists” statement of the theorem).
Optimality of Huffman

... proof of lemma 9.6.1.

With this swap, since $L(C_m)$ is minimal, we have

$$0 \leq L(C'_m) - L(C_m) = \sum_i p_i \ell'_i - \sum_i p_i \ell_i$$

(9.45)

$$= p_j \ell'_j + p_k \ell'_k - p_j \ell_j - p_k \ell_k$$

(9.46)

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k$$

(9.47)

$$= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j)$$

(9.48)

$$= (p_j - p_k) (\ell_k - \ell_j) \geq 0$$

(9.49)

Thus, $\ell'_k > \ell_j$ when $p_j > p_k$ and the code satisfies property 1.

In fact, this property is true for all optimal codes (stronger than the "there exists" statement of the theorem).
proof of lemma 9.6.1.

With this swap, since $L(C_m)$ is minimal, we have

$$0 \leq L(C'_m) - L(C_m) = \sum_i p_i \ell'_i - \sum_i p_i \ell_i \quad (9.45)$$

$$= p_j \ell'_j + p_k \ell'_k - p_j \ell_j - p_k \ell_k \quad (9.46)$$

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k \quad (9.47)$$

$$= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j) \quad (9.48)$$

$$= (p_j - p_k) (\ell_k - \ell_j) \geq 0 \quad (9.49)$$

Thus, $\ell_k \geq \ell_j$ when $p_j > p_k$ and the code satisfies property 1.
Optimality of Huffman

... proof of lemma 9.6.1.

- With this swap, since $L(C_m')$ is minimal, we have

$$0 \leq L(C_m') - L(C_m) = \sum_i p_i \ell_i' - \sum_i p_i \ell_i$$  \hspace{0.5cm} (9.45)

$$= p_j \ell_j' + p_k \ell_k' - p_j \ell_j - p_k \ell_k$$  \hspace{0.5cm} (9.46)

$$= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k$$  \hspace{0.5cm} (9.47)

$$= p_j (\ell_k - \ell_j) - p_k (\ell_k - \ell_j)$$  \hspace{0.5cm} (9.48)

$$= (p_j - p_k) (\ell_k - \ell_j) \geq 0$$  \hspace{0.5cm} (9.49)

- Thus, $\ell_k \geq \ell_j$ when $p_j > p_k$ and the code satisfies property 1.

- In fact, this property is true for all optimal codes (stronger than the “there exists” statement of the theorem).

...
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 2 (longest codewords have the same length).

...
proof of lemma 9.6.1.

- Property 2 (longest codewords have the same length).
- If two longest codewords are not the same length, then delete the last bit of the longer one.
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 2 (longest codewords have the same length).

- If two longest codewords are not the same length, then delete the last bit of the longer one. ⇒ we retain the prefix property, since the longest codeword is unique in its length has no prefix that is a code.
... proof of lemma 9.6.1.

- Property 2 (longest codewords have the same length).
- If two longest codewords are not the same length, then delete the last bit of the longer one. \(\Rightarrow\) we retain the prefix property, since the longest codeword is unique in its length has no prefix that is a code.

if siblings after deletion

if not siblings after deletion

\[
\begin{align*}
\text{...}
\end{align*}
\]
Optimality of Huffman

...proof of lemma 9.6.1.

- Property 2 (longest codewords have the same length).

- If two longest codewords are not the same length, then delete the last bit of the longer one. \(\Rightarrow\) we retain the prefix property, since the longest codeword is unique in its length has no prefix that is a code.
  - if siblings after deletion
  - if not siblings after deletion

\[\Rightarrow\] we have reduced expected length.

\[\ldots\]
... proof of lemma 9.6.1.

- Property 2 (longest codewords have the same length).
- If two longest codewords are not the same length, then delete the last bit of the longer one. \( \Rightarrow \) we retain the prefix property, since the longest codeword is unique in its length has no prefix that is a code.
  - if siblings after deletion
  - if not siblings after deletion

\( \Rightarrow \) we have reduced expected length. \( \Rightarrow \) optimal code must have two longest codewords with the same length.
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 3 (two longest codewords differ only in last bit & correspond to two least likely source symbols).

Due to property 1 ($p_k < p_j$), if $p_k$ is the smallest probability, then it must have a codeword length no less than any other $j$ with $p_j > p_k$. Similarly, if $p_k$ is second least probable, then it has codeword length no less than any more probable symbol.

Thus, the two longest codewords have the same length (prop 2) and correspond to two least likely source symbols. If the two longest codewords are not siblings, we can swap them. I.e., if $p_1 \leq p_2 \leq \cdots \leq p_m$ then do the transformation:

...
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 3 (two longest codewords differ only in last bit & correspond to two least likely source symbols).

- Due to property 1 \( (p_k < p_j \Rightarrow \ell_k \geq \ell_j) \), if \( p_k \) is the smallest probability, then it must have a codeword length no less than any other \( j \) with \( p_j > p_k \). Similarly, if \( p_k \) is second least probable, then it has codeword length no less than any more probable symbol.
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 3 (two longest codewords differ only in last bit & correspond to two least likely source symbols).
- Due to property 1 \((p_k < p_j \Rightarrow \ell_k \geq \ell_j)\), if \(p_k\) is the smallest probability, then it must have a codeword length no less than any other \(j\) with \(p_j > p_k\). Similarly, if \(p_k\) is second least probable, then it has codeword length no less than any more probable symbol.
- Thus, the two longest codewords have same length (prop 2) and correspond to two least likely source symbols.

...
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 3 (two longest codewords differ only in last bit & correspond to two least likely source symbols).
- Due to property 1 \( (p_k < p_j \Rightarrow \ell_k \geq \ell_j) \), if \( p_k \) is the smallest probability, then it must have a codeword length no less than any other \( j \) with \( p_j > p_k \). Similarly, if \( p_k \) is second least probable, then it has codeword length no less than any more probable symbol.
- Thus, the two longest codewords have same length (prop 2) and correspond to two least likely source symbols.
- If the two longest codewords are not siblings, we can swap them. I.e., if \( p_1 \geq p_2 \geq \cdots \geq p_m \) then do the transformation:

...
Optimality of Huffman

... proof of lemma 9.6.1.

- Property 3 (two longest codewords differ only in last bit & correspond to two least likely source symbols).

- Due to property 1 \((p_k < p_j \Rightarrow \ell_k \geq \ell_j)\), if \(p_k\) is the smallest probability, then it must have a codeword length no less than any other \(j\) with \(p_j > p_k\). Similarly, if \(p_k\) is second least probable, then it has codeword length no less than any more probable symbol.

- Thus, the two longest codewords have same length (prop 2) and correspond to two least likely source symbols.

- If the two longest codewords are not siblings, we can swap them. I.e., if \(p_1 \geq p_2 \geq \cdots \geq p_m\) then do the transformation:

\[
\begin{array}{c}
\text{pm} \\
\text{pm-1}
\end{array}
\quad \longrightarrow \quad
\begin{array}{c}
\text{pm-1} \\
\text{pm}
\end{array}
\]

...
...proof of lemma 9.6.1.

- This does not change the length $L = \sum_i p_i \ell_i$. 
Optimality of Huffman

...proof of lemma 9.6.1.

- This does not change the length \( L = \sum_i p_i \ell_i \).

- Thus, if \( p_1 \geq p_2 \geq \cdots \geq p_m \), there exists an optimal code with \( \ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1} = \ell_m \) and where \( C'(x_{m-1}) \) and \( C'(x_m) \) differ only in last bit.
Optimality of Huffman

... proof of lemma 9.6.1.

- This does not change the length \[ L = \sum_i p_i \ell_i. \]

- Thus, if \( p_1 \geq p_2 \geq \cdots \geq p_m \), there exists an optimal code with \( \ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1} = \ell_m \) and where \( C'(x_{m-1}) \) and \( C'(x_m) \) differ only in last bit.

- So, next we’re going to demonstrate how Huffman is optimal by starting with a code and then doing a Huffman operation to produce a new code, and where the optimization of the original code is dependent on a (simpler) optimization on a shorter code.
Optimality of Huffman

... proof of lemma 9.6.1.

- This does not change the length \( L = \sum_i p_i \ell_i \).

- Thus, if \( p_1 \geq p_2 \geq \cdots \geq p_m \), there exists an optimal code with \( \ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1} = \ell_m \) and where \( C(x_{m-1}) \) and \( C(x_m) \) differ only in last bit.

- So, next we’re going to demonstrate how Huffman is optimal by starting with a code and then doing a Huffman operation to produce a new code, and where the optimization of the original code is dependent on a (simpler) optimization on a shorter code.

- We’ll continue doing this until the optimal code will be apparent.
Optimality of Huffman

...proof of lemma 9.6.1.

This does not change the length $L = \sum_i p_i \ell_i$.

Thus, if $p_1 \geq p_2 \geq \cdots \geq p_m$, there exists an optimal code with $
\ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1} = \ell_m$ and where $C(x_{m-1})$ and $C(x_m)$ differ only in last bit.

So, next we’re going to demonstrate how Huffman is optimal by starting with a code and then doing a Huffman operation to produce a new code, and where the optimization of the original code is dependent on a (simpler) optimization on a shorter code.

We’ll continue doing this until the optimal code will be apparent.

Assume (some not necessarily optimal) code $C_m$ (on $m$ symbols) that satisfies the above properties. $C_m$ has codewords $\{\omega_i\}_{i=1}^m$. 
Optimality of Huffman

...proof of lemma 9.6.1.

- This does not change the length \( L = \sum_i p_i \ell_i \).

- Thus, if \( p_1 \geq p_2 \geq \cdots \geq p_m \), there exists an optimal code with \( \ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1} = \ell_m \) and where \( C(x_{m-1}) \) and \( C(x_m) \) differ only in last bit.

- So, next we’re going to demonstrate how Huffman is optimal by starting with a code and then doing a Huffman operation to produce a new code, and where the optimization of the original code is dependent on a (simpler) optimization on a shorter code.

- We’ll continue doing this until the optimal code will be apparent.

- Assume (some not necessarily optimal) code \( C_m \) (on \( m \) symbols) that satisfies the above properties. \( C_m \) has codewords \( \{\omega_i\}_{i=1}^m \).

- Huffman turns code \( C_m \) into code \( C_{m-1} \) (with codewords \( \{\omega'_i\}_{i=1}^{m-1} \) ).
Assume (some not necessarily optimal) code $C_m$ (on $m$ symbols) that satisfies the above properties. $C_m$ has codewords $\{\omega_i\}_{i=1}^m$. 

Hu builds the code backwards, taking the two smallest probabilities $p_{m1}, p_{m2}$, giving a bit (0 or 1) to each code word, and merges passing the result back to another round of Hu.
Assume (some not necessarily optimal) code $C_m$ (on $m$ symbols) that satisfies the above properties. $C_m$ has codewords $\{w_i\}_{i=1}^m$.

Indices $m, m-1$ have the least probability and longest codewords.

<table>
<thead>
<tr>
<th>$C_m$</th>
<th>length</th>
<th>symb. prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$l_1$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$w_2$</td>
<td>$l_2$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$w_{m-2}$</td>
<td>$l_{m-2}$</td>
<td>$p_{m-2}$</td>
</tr>
<tr>
<td>$w_{m-1}$</td>
<td>$l_{m-1}$</td>
<td>$p_{m-1}$</td>
</tr>
<tr>
<td>$w_m$</td>
<td>$l_m$</td>
<td>$p_m$</td>
</tr>
</tbody>
</table>
Optimality of Huffman

- Assume (some not necessarily optimal) code $C_m$ (on $m$ symbols) that satisfies the above properties. $C_m$ has codewords $\{\omega_i\}_{i=1}^m$.

- Indices $m, m-1$ have the least probability and longest codewords.

<table>
<thead>
<tr>
<th>$C_m$</th>
<th>length</th>
<th>symb. prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$l_1$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$w_2$</td>
<td>$l_2$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$w_{m-2}$</td>
<td>$l_{m-2}$</td>
<td>$p_{m-2}$</td>
</tr>
<tr>
<td>$w_{m-1}$</td>
<td>$l_{m-1}$</td>
<td>$p_{m-1}$</td>
</tr>
<tr>
<td>$w_m$</td>
<td>$l_m$</td>
<td>$p_m$</td>
</tr>
</tbody>
</table>

- Huffman builds the code backwards, taking the two smallest probabilities $p_{m-1}, p_m$, giving a bit (0 or 1) to each code word, and merges passing the result back to another round of Huffman.
Huffman implicitly goes from current code $C_m$ to $C_{m-1}$ as follows:

<table>
<thead>
<tr>
<th>symb. prob.</th>
<th>$C_{m-1}$</th>
<th>$m-1$ len.</th>
<th>code rel.</th>
<th>length relationship</th>
<th>symb. prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$\omega'_1$</td>
<td>$\ell'_1$</td>
<td>$w_1 = w'_1$</td>
<td>$\ell_1 = \ell'_1$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$\omega'_2$</td>
<td>$\ell'_2$</td>
<td>$w_2 = w'_2$</td>
<td>$\ell_2 = \ell'_2$</td>
<td>$p_2$</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$p_{m-2}$</td>
<td>$\omega'_{m-2}$</td>
<td>$\ell'_{m-2}$</td>
<td>$w_{m-2} = w'_{m-2}$</td>
<td>$\ell_{m-2} = \ell'_{m-2}$</td>
<td>$p_{m-2}$</td>
</tr>
<tr>
<td>$p_{m-1} + p_m$</td>
<td>$\omega'_{m-1}$</td>
<td>$\ell'_{m-1}$</td>
<td>$w_{m-1} = w'_{m-1}$</td>
<td>$\ell_{m-1} = \ell'_{m-1} + 1$</td>
<td>$p_{m-1}$</td>
</tr>
<tr>
<td></td>
<td>$w_m = w'_{m-1}$</td>
<td>$\ell_m = \ell'_{m-1} + 1$</td>
<td>$p_m$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Huffman implicitly goes from current code $C_m$ to $C_{m-1}$ as follows:

<table>
<thead>
<tr>
<th>symb. prob.</th>
<th>$C_{m-1}$</th>
<th>$m-1$ len.</th>
<th>code rel.</th>
<th>length relationship</th>
<th>symb. prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$\omega_1'$</td>
<td>$\ell_1'$</td>
<td>$w_1 = w_1'$</td>
<td>$\ell_1 = \ell_1'$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$\omega_2'$</td>
<td>$\ell_2'$</td>
<td>$w_2 = w_2'$</td>
<td>$\ell_2 = \ell_2'$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$p_{m-2}$</td>
<td>$\omega_{m-2}'$</td>
<td>$\ell_{m-2}'$</td>
<td>$w_{m-2} = w_{m-2}'$</td>
<td>$\ell_{m-2} = \ell_{m-2}'$</td>
<td>$p_{m-2}$</td>
</tr>
<tr>
<td>$p_{m-1} + p_m$</td>
<td>$\omega_{m-1}'$</td>
<td>$\ell_{m-1}'$</td>
<td>$w_{m-1} = w_{m-1}'$</td>
<td>$\ell_{m-1} = \ell_{m-1}' + 1$</td>
<td>$p_{m-1}$</td>
</tr>
<tr>
<td></td>
<td>$\omega_{m}'$</td>
<td>$\ell_{m}'$</td>
<td>$w_{m} = w_{m}'$</td>
<td>$\ell_{m} = \ell_{m}' + 1$</td>
<td>$p_m$</td>
</tr>
</tbody>
</table>

Again, $\omega_i$ are the $C_m$ lengths and $\omega_i'$ are the $C_{m-1}$ lengths.
### Optimality of Huffman

- Huffman implicitly goes from current code $C_m$ to $C_{m-1}$ as follows:

<table>
<thead>
<tr>
<th>symb. prob.</th>
<th>$C_{m-1}$</th>
<th>$m - 1$ len.</th>
<th>code rel.</th>
<th>length relationship</th>
<th>symb. prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$\omega'_1$</td>
<td>$\ell'_1$</td>
<td>$w_1 = w'_1$</td>
<td>$\ell_1 = \ell'_1$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$\omega'_2$</td>
<td>$\ell'_2$</td>
<td>$w_2 = w'_2$</td>
<td>$\ell_2 = \ell'_2$</td>
<td>$p_2$</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$p_{m-2}$</td>
<td>$\omega'_{m-2}$</td>
<td>$\ell'_{m-2}$</td>
<td>$w_{m-2} = w'_{m-2}$</td>
<td>$\ell_{m-2} = \ell'_{m-2}$</td>
<td>$p_{m-2}$</td>
</tr>
<tr>
<td>$p_{m-1} + p_m$</td>
<td>$\omega'_{m-1}$</td>
<td>$\ell'_{m-1}$</td>
<td>$w_{m-1} = w'_{m-1}$</td>
<td>$\ell_{m-1} = \ell'_{m-1} + 1$</td>
<td>$p_{m-1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$w_m = w'_{m-1}$</td>
<td>$\ell_m = \ell'_{m-1} + 1$</td>
<td></td>
<td>$p_m$</td>
</tr>
</tbody>
</table>

- Again, $\omega_i$ are the $C_m$ lengths and $\omega'_i$ are the $C_{m-1}$ lengths.

- Lengths are defined recursively at the time of the Huffman step. All Huffman knows is the relationship between the current lengths and codewords (at step $m$) to the next lengths and codewords (at step $m - 1$). Huffman is lazy in this way.
Optimality of Huffman

- We get the following:

\[ L(C_m) = \sum_{i=1}^{m} p_i l_i \]

\[ = m_2 \sum_{i=1}^{m} p_i l_i + m_1 \]  

\[ = \sum_{i=1}^{m} p_0 l_i + (p_1 + p_2) m_1 + p_2 \]

\[ = L(C_{m_1}) + p_1 + p_2 \]

(9.54)
Optimality of Huffman

We get the following:

\[ L(C_m) \]

(9.54)
Optimality of Huffman

We get the following:

\[ L(C_m) = \sum_i p_i \ell_i \]  (9.50)

\[ L(C_m) = \sum_i p_i \ell_i \]  (9.54)

Reduces num. of variables we need to optimize over.
Optimality of Huffman

We get the following:

\[
L(C_m) = \sum_i p_i \ell_i \tag{9.50}
\]

\[
= \sum_{i=1}^{m-2} p_i \ell_i' + p_{m-1}(\ell_{m-1}' + 1) + p_m(\ell_{m-1}' + 1) \tag{9.51}
\]

\[
= L(C_{m-1}) + p_{m-1} + p_m | \{z\} \tag{9.54}
\]
Optimality of Huffman

- We get the following:

\[
L(C_m) = \sum_{i} p_i \ell_i
\]  \hspace{1cm} (9.50)

\[
= \sum_{i=1}^{m-2} p_i \ell_i' + p_{m-1} (\ell_{m-1}' + 1) + p_m (\ell_{m-1}' + 1)
\]  \hspace{1cm} (9.51)

\[
= \sum_{i=1}^{m-2} p_i \ell_i' + (p_{m-1} + p_m) \ell_{m-1}' + p_{m-1} + p_m
\]  \hspace{1cm} (9.52)

(9.54)
Optimality of Huffman

- We get the following:

\[
L(C_m) = \sum_i p_i \ell_i
\]

(9.50)

\[
= \sum_{i=1}^{m-2} p_i \ell'_i + p_{m-1}(\ell'_{m-1} + 1) + p_m(\ell'_{m-1} + 1)
\]

(9.51)

\[
= \sum_{i=1}^{m-2} p_i \ell'_i + (p_{m-1} + p_m)\ell'_{m-1} + p_{m-1} + p_m
\]

(9.52)

\[
= \sum_{i=1}^{m-1} p'_i \ell'_i + p_{m-1} + p_m
\]

(9.53)

\[
= p_m + p_m
\]

(9.54)
Optimality of Huffman

We get the following:

\[ L(C_m) = \sum_{i} p_i \ell_i \]  
\[ = \sum_{i=1}^{m-2} p_i \ell_i' + p_{m-1}(\ell_{m-1}' + 1) + p_m(\ell_{m-1}' + 1) \]  
\[ = \sum_{i=1}^{m-2} p_i \ell_i' + (p_{m-1} + p_m)\ell_{m-1}' + p_{m-1} + p_m \]  
\[ = \sum_{i=1}^{m-1} p_i' \ell_i' + p_{m-1} + p_m \]  
\[ = L(C_{m-1}) + p_{m-1} + p_m \]
Optimality of Huffman

- We get the following:

\[ L(C_m) = \sum_{i} p_i \ell_i \]  \hspace{1cm} (9.50)

\[ = \sum_{i=1}^{m-2} p_i \ell'_i + p_{m-1}(\ell'_{m-1} + 1) + p_m(\ell'_{m-1} + 1) \]  \hspace{1cm} (9.51)

\[ = \sum_{i=1}^{m-2} p_i \ell'_i + (p_{m-1} + p_m)\ell'_{m-1} + p_m - 1 + p_m \]  \hspace{1cm} (9.52)

\[ = \sum_{i=1}^{m-1} p'_i \ell'_i + p_{m-1} + p_m \]  \hspace{1cm} (9.53)

\[ = L(C_{m-1}) + \underbrace{p_{m-1} + p_m}_{\text{doesn't involve lengths}} \]  \hspace{1cm} (9.54)
Optimality of Huffman

- We get the following:

\[
L(C_m) = \sum_i p_i \ell_i
\]  
(9.50)

\[
= \sum_{i=1}^{m-2} p_i \ell'_i + p_{m-1}(\ell'_{m-1} + 1) + p_m(\ell'_{m-1} + 1)
\]  
(9.51)

\[
= \sum_{i=1}^{m-2} p_i \ell'_i + (p_{m-1} + p_m)\ell'_{m-1} + p_{m-1} + p_m
\]  
(9.52)

\[
= \sum_{i=1}^{m-1} p'_i \ell'_i + p_{m-1} + p_m
\]  
(9.53)

\[
= L(C_{m-1}) + p_{m-1} + p_m
\]  
(9.54)

- Reduces num. of variables we need to optimize over.
Optimality of Huffman

So the Huffman procedure implies that:

\[
\min_{\ell_1:m} L(C_m) = \text{const.} + \min_{\ell_1:m-1} L(C_{m-1}) = \ldots
\]

\[
= \text{const.} + \min_{\ell_1:2} L(C_2)
\]

where each min step is Huffman, and each preserves the stated properties.
Optimality of Huffman

- So the Huffman procedure implies that:

\[
\min_{\ell_1:m} L(C_m) = \text{const.} + \min_{\ell_1:m-1} L(C_{m-1}) = \ldots \tag{9.55}
\]

\[
= \text{const.} + \min_{\ell_1:2} L(C_2) \tag{9.56}
\]

where each \( \min \) step is Huffman, and each preserves the stated properties.

- This reduces down to a length-2 code, which is obvious to optimize (use one bit for each source symbol), and then we backtrack to construct the code.
Optimality of Huffman

- So the Huffman procedure implies that:

\[
\min_{\ell_1:\ell_m} L(C_{m}) = \text{const.} + \min_{\ell_1:\ell_{m-1}} L(C_{m-1}) = \ldots 
\]

\[
= \text{const.} + \min_{\ell_1:\ell_2} L(C_{2})
\]

(9.55)

(9.56)

where each min step is Huffman, and each preserves the stated properties.

- This reduces down to a length-2 code, which is obvious to optimize (use one bit for each source symbol), and then we backtrack to construct the code.

- Optimality is preserved at each step. We kept the properties of the code, and reduced the problem to one having only one (obvious) solution.
Optimality of Huffman

- So the Huffman procedure implies that:

\[
\min_{\ell_1:m} L(C_m) = \text{const.} + \min_{\ell_1:m-1} L(C_{m-1}) = \ldots \tag{9.55}
\]

\[
= \text{const.} + \min_{\ell_1:2} L(C_2) \tag{9.56}
\]

where each min step is Huffman, and each preserves the stated properties.

- This reduces down to a length-2 code, which is obvious to optimize (use one bit for each source symbol), and then we backtrack to construct the code.

- Optimality is preserved at each step. We kept the properties of the code, and reduced the problem to one having only one (obvious) solution.

**Theorem 9.6.2**

*The Huffman coding procedure is an optimal integer code lengths code.*
Huffman coding is a symbol code, we code one symbol at a time.
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal?
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? **But what does optimal mean?**
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? But what does optimal mean?
- In general, for a symbol code, each symbol in the source alphabet must use an integer number of codeword bits.
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? But what does optimal mean?
- In general, for a symbol code, each symbol in the source alphabet must use an integer number of codeword bits.
- This is ok for $D$-adic distributions but could use up to one extra bit per symbol on average.
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? But what does optimal mean?
- In general, for a symbol code, each symbol in the source alphabet must use an integer number of codeword bits.
- This is ok for $D$-adic distributions but could use up to one extra bit per symbol on average.
- Bad example: $p(0) = 1 - p(1) = 0.999$, then $-\log p(0) \approx 0$, so we should be using close to zero bits per symbol to code this, but Huffman uses 1.
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? But what does optimal mean?
- In general, for a symbol code, each symbol in the source alphabet must use an integer number of codeword bits.
- This is ok for $D$-adic distributions but could use up to one extra bit per symbol on average.
- Bad example: $p(0) = 1 - p(1) = 0.999$, then $-\log p(0) \approx 0$, so we should be using close to zero bits per symbol to code this, but Huffman uses 1.
- Thus, we need a long block to get any benefit.
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? But what does optimal mean?
- In general, for a symbol code, each symbol in the source alphabet must use an integer number of codeword bits.
- This is ok for $D$-adic distributions but could use up to one extra bit per symbol on average.
- Bad example: $p(0) = 1 - p(1) = 0.999$, then $-\log p(0) \approx 0$, so we should be using close to zero bits per symbol to code this, but Huffman uses 1.
- Thus, we need a long block to get any benefit.
- In practice, this means we need to store and be able to compute $p(x_1:n)$. 
Huffman Codes

- Huffman coding is a symbol code, we code one symbol at a time.
- Is Huffman optimal? But what does optimal mean?
- In general, for a symbol code, each symbol in the source alphabet must use an integer number of codeword bits.
- This is ok for $D$-adic distributions but could use up to one extra bit per symbol on average.
- Bad example: $p(0) = 1 - p(1) = 0.999$, then $-\log p(0) \approx 0$, so we should be using close to zero bits per symbol to code this, but Huffman uses 1.
- Thus, we need a long block to get any benefit.
- In practice, this means we need to store and be able to compute $p(x_{1:n})$. No problem, right?
Can we easily compute $p(x_1:n)$?

If $|A|$ is the alphabet size, we need a table of size $|A|^n$ to store these probabilities. Moreover, it is hard to estimate $p(x_1:n)$ accurately. Given an amount of "training data" (to borrow a phrase from machine learning), it is hard to estimate this distribution. Many of the possible strings in any finite sample size will not occur (sparsity).

Example: how hard is it to find a short grammatically valid English phrase never before written using a web search engine?

"dogs ate banks on the river" is not found as of Mon, Oct 28, 2013. Smoothing models are required. Similar to the language model problem in natural language processing.
Huffman Codes

- Can we easily compute $p(x_{1:n})$?

- If $|A|$ is the alphabet size, we need a table of size $|A|^n$ to store these probabilities.
Can we easily compute $p(x_{1:n})$?

If $|A|$ is the alphabet size, we need a table of size $|A|^n$ to store these probabilities.

Moreover, it is hard to estimate $p(x_{1:n})$ accurately. Given an amount of “training data” (to borrow a phrase from machine learning), it is hard to estimate this distribution. Many of the possible strings in any finite sample size will not occur (sparsity).
Can we easily compute \( p(x_{1:n}) \)?

If \(|A|\) is the alphabet size, we need a table of size \(|A|^n\) to store these probabilities.

Moreover, it is hard to estimate \( p(x_{1:n}) \) accurately. Given an amount of “training data” (to borrow a phrase from machine learning), it is hard to estimate this distribution. Many of the possible strings in any finite sample size will not occur (sparsity).

Example: how hard is it to find a short grammatically valid English phrase never before written using a web search engine?
Huffman Codes

- Can we easily compute $p(x_{1:n})$?
- If $|A|$ is the alphabet size, we need a table of size $|A|^n$ to store these probabilities.
- Moreover, it is hard to estimate $p(x_{1:n})$ accurately. Given an amount of “training data” (to borrow a phrase from machine learning), it is hard to estimate this distribution. Many of the possible strings in any finite sample size will not occur (sparsity).
- Example: how hard is it to find a short grammatically valid English phrase never before written using a web search engine? “dogs ate banks on the river” is not found as of Mon, Oct 28, 2013.
Huffman Codes

- Can we easily compute \( p(x_1:n) \)?
- If \( |A| \) is the alphabet size, we need a table of size \( |A|^n \) to store these probabilities.
- Moreover, it is hard to estimate \( p(x_1:n) \) accurately. Given an amount of “training data” (to borrow a phrase from machine learning), it is hard to estimate this distribution. Many of the possible strings in any finite sample size will not occur (sparsity).
- Example: how hard is it to find a short grammatically valid English phrase never before written using a web search engine? “dogs ate banks on the river” is not found as of Mon, Oct 28, 2013.
- Smoothing models are required. Similar to the language model problem in natural language processing.
Huffman Codes

- Huffman has the property that

\[ H(X) \leq L(\text{Huffman}) \leq H(X) + 1 \]  \hspace{1cm} (9.57)
Huffman Codes

- Huffman has the property that

\[ H(X) \leq L(\text{Huffman}) \leq H(X) + 1 \]  \hspace{1cm} (9.57)

- Bigger block sizes help, but we get

\[ H(X_{1:n}) \leq L(\text{Block Huffman}) \leq H(X_{1:n}) + 1 \]  \hspace{1cm} (9.58)

for the block.
Huffman Codes

- Huffman has the property that

\[ H(X) \leq L(\text{Huffman}) \leq H(X) + 1 \] (9.57)

- Bigger block sizes help, but we get

\[ H(X_{1:n}) \leq L(\text{Block Huffman}) \leq H(X_{1:n}) + 1 \] (9.58)

for the block.

- If \( H(X_{1:n}) \) is small (e.g., English text) then this extra bit can be significant.
Huffman Codes

- Huffman has the property that

\[ H(X) \leq L(\text{Huffman}) \leq H(X) + 1 \]  \hspace{1cm} (9.57)

- Bigger block sizes help, but we get

\[ H(X_{1:n}) \leq L(\text{Block Huffman}) \leq H(X_{1:n}) + 1 \]  \hspace{1cm} (9.58)

for the block.

- If \( H(X_{1:n}) \) is small (e.g., English text) then this extra bit can be significant.

- If block gets too long, we have the estimation problem again (hard to compute \( p(x_{1:n}) \),
Huffman Codes

- Huffman has the property that

\[ H(X) \leq L(\text{Huffman}) \leq H(X) + 1 \]  \hspace{1cm} \text{(9.57)}

- Bigger block sizes help, but we get

\[ H(X_{1:n}) \leq L(\text{Block Huffman}) \leq H(X_{1:n}) + 1 \]  \hspace{1cm} \text{(9.58)}

for the block.

- If \( H(X_{1:n}) \) is small (e.g., English text) then this extra bit can be significant.

- If block gets too long, we have the estimation problem again (hard to compute \( p(x_{1:n}) \),

- also the fact that it introduces latencies (we need to encode and then wait for the end of a block before we can send any bits).