Read chapters 1, and 2 in C&T.
Read chapter 3 in C&T.
Read section 11.1,11.3, method of types and universal source coding.
Read chapter 4.
Read chapter 5.
Reminder: Homework 3 due this Sunday (Feb 5th) at 5:00pm our dropbox (https://catalyst.uw.edu/collectit/dropbox/karna/19164).

This homework is long! Don’t wait to start it!

Late policy: 10% every 24 hour period that you are late, and no more than 3 days late accepted.

Lowest grade out of all HW grades is not counted towards final grade (so you can skip one HW with impunity).

Please do use our discussion board (https://catalyst.uw.edu/gopost/board/karna/25503/) for all questions, so that all will benefit from them being answered.
Class Road Map

- L1 (1/3): Overview, Entropy
- L2 (1/5): Props. Entropy, Mutual Information, KL-Divergence
- L3 (1/10): KL-Divergence, Jensen, properties, Data Proc. Inequality
- L5 (1/17): Fano, AEP
- L6 (1/19): snow
- L6 (1/24): AEP, source coding
- L7 (1/26): Method of Types
- L9 (2/2): HMMs, coding
- L10 (2/7): Coding,
- L11 (2/9): midterm discussion
- L12 (2/14): Midterm
- L13 (2/16): 
- L14 (2/21): 
- L15 (2/23): 
- L16 (2/28): 
- L17 (3/1): 
- L18 (3/6): 
- L19 (3/8): 

Finals Week: March 12th–16th.
Universal Source Coding

If we know \( p(x) \), then we will be able to develop a code to compress sources generated by \( p(x) \). Huffman, Lempel-Ziv, etc. are codes that, as we will soon see, do that.
Universal Source Coding

- If we know $p(x)$, then we will be able to develop a code to compress sources generated by $p(x)$. Huffman, Lempel-Ziv, etc. are codes that, as we will soon see, do that.

- What if we don't know $p(x)$?
Universal Source Coding

- If we know $p(x)$, then we will be able to develop a code to compress sources generated by $p(x)$. Huffman, Lempel-Ziv, etc. are codes that, as we will soon see, do that.

- What if we don't know $p(x)$?

- Q: do there exist codes that can compress without knowing $p(x)$ and that do so down to the entropy limit?
Universal Source Coding

- If we know $p(x)$, then we will be able to develop a code to compress sources generated by $p(x)$. Huffman, Lempel-Ziv, etc. are codes that, as we will soon see, do that.
- What if we don't know $p(x)$?
- Q: do there exist codes that can compress without knowing $p(x)$ and that do so down to the entropy limit?
- Q: can we compress down to the rate $R$ (in units of bits per source symbol) if $R > H(Q)$? (this is Shannon's source coding theorem)
Universal Source Coding

- If we know $p(x)$, then we will be able to develop a code to compress sources generated by $p(x)$. Huffman, Lempel-Ziv, etc. are codes that, as we will soon see, do that.

- What if we don't know $p(x)$?

- Q: do there exist codes that can compress without knowing $p(x)$ and that do so down to the entropy limit?

- Q: can we compress down to the rate $R$ (in units of bits per source symbol) if $R > H(Q)$? (this is Shannon’s source coding theorem)

- What happens if $R < H(Q)$? (this is the converse of Shannon’s source coding theorem)
Universal Source Coding

- If we know $p(x)$, then we will be able to develop a code to compress sources generated by $p(x)$. Huffman, Lempel-Ziv, etc. are codes that, as we will soon see, do that.
- What if we don't know $p(x)$?
- Q: do there exist codes that can compress without knowing $p(x)$ and that do so down to the entropy limit?
- Q: can we compress down to the rate $R$ (in units of bits per source symbol) if $R > H(Q)$? (this is Shannon’s source coding theorem)
- What happens if $R < H(Q)$? (this is the converse of Shannon’s source coding theorem)
- We’ll formally prove this theorem using the method of types.
Fixed rate block code of rate $R$

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
Fixed rate block code of rate $R$

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
Fixed rate block code of rate $R$

- An $(M,n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
- Then a code is defined as follows:
Fixed rate block code of rate $R$

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
- Then a code is defined as follows:

**Definition 2.1 (fixed rate block code of rate $R$)**

Let $X_1, X_2, \ldots, X_n \sim Q$, i.i.d. but $Q$ unknown. We have encoder and decoder functions as follows:

$$f_n: X_n \to \{1, 2, \ldots, 2^{nR}\}$$

$$\phi_n: \{1, 2, \ldots, 2^{nR}\} \to X_n$$

and probability of error $P_e(n) = Q_n(\{x_1:n: \phi_n(f_n(x_1:n)) \neq x_1:n\})$
Fixed rate block code of rate $R$

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
- Then a code is defined as follows:

**Definition 2.1 (fixed rate block code of rate $R$)**

Let $X_1, X_2, \ldots, X_n \sim Q$, i.i.d. but $Q$ unknown. We have encoder and decoder functions as follows:

Encoder: $f_n : \mathcal{X}^n \rightarrow \{1, 2, \ldots, 2^{nR}\}$ \hspace{1cm} (1)

Decoder: $\phi_n : \{1, 2, \ldots, 2^{nR}\} \rightarrow X^n$ \hspace{1cm} (2)
**Fixed rate block code of rate $R$**

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
- Then a code is defined as follows:

**Definition 2.1 (fixed rate block code of rate $R$)**

Let $X_1, X_2, \ldots, X_n \sim Q$, i.i.d. but $Q$ unknown. We have encoder and decoder functions as follows:

Encoder: $f_n : \mathcal{X}^n \to \{1, 2, \ldots, 2^{nR}\}$  \hspace{1cm} (1)

Decoder: $\phi_n : \{1, 2, \ldots, 2^{nR}\} \to \mathcal{X}^n$  \hspace{1cm} (2)
Fixed rate block code of rate $R$

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
- Then a code is defined as follows:

**Definition 2.1 (fixed rate block code of rate $R$)**

Let $X_1, X_2, \ldots, X_n \sim Q$, i.i.d. but $Q$ unknown. We have encoder and decoder functions as follows:

Encoder: $f_n : X^n \rightarrow \{1, 2, \ldots, 2^{nR}\}$ \hspace{1cm} (1)

Decoder: $\phi_n : \{1, 2, \ldots, 2^{nR}\} \rightarrow X^n$ \hspace{1cm} (2)

and probability of error

$$P_e^{(n)} = Q^n(\{x_{1:n} : \phi_n(f_n(x_{1:n})) \neq x_{1:n}\})$$ \hspace{1cm} (3)
Fixed rate block code of rate $R$

- An $(M, n)$ code is one that uses $M$ code words for $n$ source symbols.
- Such a code thus has rate of $R = \frac{\log M}{n}$ bits per source symbol. So we need $\log M = nR$ bits to index this code.
- Then a code is defined as follows:

**Definition 2.1 (fixed rate block code of rate $R$)**

Let $X_1, X_2, \ldots, X_n \sim Q$, i.i.d. but $Q$ unknown. We have encoder and decoder functions as follows:

Encoder: $f_n : \mathcal{X}^n \rightarrow \{1, 2, \ldots, 2^{nR}\}$ \hspace{1cm} (1)

Decoder: $\phi_n : \{1, 2, \ldots, 2^{nR}\} \rightarrow \mathcal{X}^n$ \hspace{1cm} (2)

and probability of error

$$P_{e}^{(n)} = Q^n(\{x_{1:n} : \phi_n(f_n(x_{1:n})) \neq x_{1:n}\})$$ \hspace{1cm} (3)

- Notation: $(M, n) = (2^{nR}, n)$ designates a series (in $n$) of such codes.
Definition 2.2 (Universal rate \( R \) block code)

A rate \( R \) block code for a source is universal if the functions \( f_n \) and \( \phi_n \) do not depend on the source distribution \( Q \) and if

\[
P_e^{(n)} \rightarrow 0 \text{ as } n \rightarrow \infty \text{ whenever } H(Q) < R
\]  

(4)
Universal Code

Definition 2.2 (Universal rate $R$ block code)

A rate $R$ block code for a source is universal if the functions $f_n$ and $\phi_n$ do not depend on the source distribution $Q$ and if

$$P_e^{(n)} \to 0 \text{ as } n \to \infty \text{ whenever } H(Q) < R$$

So we require the “ability to code” at rate $R$, which really means code without error, or the error goes to zero for larger block length.
Universal Code

Definition 2.2 (Universal rate $R$ block code)

A rate $R$ block code for a source is universal if the functions $f_n$ and $\phi_n$ do not depend on the source distribution $Q$ and if

$$P_e(n) \to 0 \text{ as } n \to \infty \text{ whenever } H(Q) < R$$  (4)

- So we require the “ability to code” at rate $R$, which really means code without error, or the error goes to zero for larger block length.
- We next state and prove one of Shannon’s main theorems.
Universal Code

Definition 2.2 (Universal rate $R$ block code)

A rate $R$ block code for a source is universal if the functions $f_n$ and $\phi_n$ do not depend on the source distribution $Q$ and if

$$P_e^{(n)} \to 0 \text{ as } n \to \infty \text{ whenever } H(Q) < R$$

(4)

- So we require the “ability to code” at rate $R$, which really means code without error, or the error goes to zero for larger block length.
- We next state and prove one of Shannon’s main theorems.
- If $R > H(Q)$, then there exists a sequence (in $n$) of codes with the error of becoming vanishingly small.
Definition 2.2 (Universal rate $R$ block code)

A rate $R$ block code for a source is universal if the functions $f_n$ and $\phi_n$ do not depend on the source distribution $Q$ and if

$$P_e(n) \to 0 \text{ as } n \to \infty \text{ whenever } H(Q) < R \quad (4)$$

- So we require the “ability to code” at rate $R$, which really means code without error, or the error goes to zero for larger block length.
- We next state and prove one of Shannon’s main theorems.
- If $R > H(Q)$, then there exists a sequence (in $n$) of codes with the error of becoming vanishingly small.
- Conversely, if $R < H(Q)$, then the error goes to 1.
Theorem 2.3 (Shannon’s Source Coding Theorem)

\[ \exists \text{ a sequence } (2^{nR}, n) \text{ of universal source codes such that } P_e^{(n)} \to 0 \text{ for all source distributions } Q \text{ such that } H(Q) < R. \]

- Conversely, if \( R < H(Q) \) then \( P_e^{(n)} \to 1. \)
Theorem 2.3 (Shannon’s Source Coding Theorem)

There exists a sequence \((2^{nR}, n)\) of universal source codes such that \(P_e(n) \to 0\) for all source distributions \(Q\) such that \(H(Q) < R\).

- Conversely, if \(R < H(Q)\) then \(P_e(n) \to 1\).
- So entropy is a good measure of the “information” in a source.
Theorem 2.3 (Shannon’s Source Coding Theorem)

∃ a sequence \((2^nR, n)\) of universal source codes such that \(P_e(n) \to 0\) for all source distributions \(Q\) such that \(H(Q) < R\).

- Conversely, if \(R < H(Q)\) then \(P_e(n) \to 1\).
- So entropy is a good measure of the “information” in a source.
- Can you compress a compressed file? `gzip(gzip(gzip(...gzip(x))))`
Source Coding Theorem

Theorem 2.3 (Shannon’s Source Coding Theorem)

\[ \exists \text{ a sequence } (2^{nR}, n) \text{ of universal source codes such that } P_e(n) \to 0 \text{ for all source distributions } Q \text{ such that } H(Q) < R. \]

- Conversely, if \( R < H(Q) \) then \( P_e(n) \to 1 \).
- So entropy is a good measure of the “information” in a source.
- Can you compress a compressed file? \( \text{gzip(gzip(gzip(\ldots gzip(x))))} \)
- Once a source is compressed to the the entropy rate, it is uncompressible, it is at its entropy limit.
Theorem 2.3 (Shannon’s Source Coding Theorem)

∃ a sequence \((2^n R, n)\) of universal source codes such that \(P_e(n) \rightarrow 0\) for all source distributions \(Q\) such that \(H(Q) < R\).

- Conversely, if \(R < H(Q)\) then \(P_e(n) \rightarrow 1\).
- So entropy is a good measure of the “information” in a source.
- Can you compress a compressed file? gzip(gzip(gzip(...gzip(x))))
- Once a source is compressed to the the entropy rate, it is uncompressible, it is at its entropy limit.
- It looks like a random string of bits.
Source Coding Theorem

Theorem 2.3 (Shannon’s Source Coding Theorem)

∃ a sequence \((2^nR, n)\) of universal source codes such that \(P_e(n) \to 0\) for all source distributions \(Q\) such that \(H(Q) < R\).

- Conversely, if \(R < H(Q)\) then \(P_e(n) \to 1\).
- So entropy is a good measure of the “information” in a source.
- Can you compress a compressed file? \(\text{gzip(\text{gzip(\text{gzip}(...\text{gzip}(x)))))}\)
- Once a source is compressed to the the entropy rate, it is uncompressible, it is at its entropy limit.
- It looks like a random string of bits.
- How can we tell the difference? The code is the key, both figuratively and literally.
Stochastic Processes: definition brief summary

- Stationary stochastic process, statistics don’t change when we shift time.
Stochastic Processes: definition brief summary

- Stationary stochastic process, statistics don’t change when we shift time.
- Markov process (model): future and past independent given the present, or immediate past is sufficient to render more distant past irrelevant.
Stochastic Processes: definition brief summary

- Stationary stochastic process, statistics don’t change when we shift time.
- Markov process (model): future and past independent given the present, or immediate past is sufficient to render more distant past irrelevant.
- (time) homogeneous: parameters of Markov chain don’t change with time.
Stochastic Processes: definition brief summary

- Stationary stochastic process, statistics don’t change when we shift time.
- Markov process (model): future and past independent given the present, or immediate past is sufficient to render more distant past irrelevant.
- (time) homogeneous: parameters of Markov chain don’t change with time.
- Irreducible: it is possible to get from any state to any other state eventually.
Stochastic Processes: definition brief summary

- Stationary stochastic process, statistics don’t change when we shift time.
- Markov process (model): future and past independent given the present, or immediate past is sufficient to render more distant past irrelevant.
- (time) homogeneous: parameters of Markov chain don’t change with time.
- Irreducible: it is possible to get from any state to any other state eventually.
- Periodic: greatest common divisor of the time intervals where return is possible is $> 1$. 

More facts about Markov chains and stochastic processes: Great source is the text: see "Probability and Random Processes", Grimmett and Stirzaker.
Stochastic Processes: definition brief summary

- Stationary stochastic process, statistics don’t change when we shift time.
- Markov process (model): future and past independent given the present, or immediate past is sufficient to render more distant past irrelevant.
- (time) homogeneous: parameters of Markov chain don’t change with time.
- Irreducible: it is possible to get from any state to any other state eventually.
- Periodic: greatest common divisor of the time intervals where return is possible is > 1.

More facts about Markov chains and stochastic processes: Great source is the text: see “Probability and Random Processes”, Grimmett and Stirzaker.
Entropy rates

- Two definitions of entropy rates.
Entropy rates

- Two definitions of entropy rates.

Definition 2.4

The entropy rate of a stochastic process \( \{X_i\}_i \) is defined as

\[
H(\mathcal{X}) \triangleq \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n) \tag{5}
\]

when it exists.
Entropy rates

Two definitions of entropy rates.

**Definition 2.4**

The entropy rate of a stochastic process \( \{X_i\}_i \) is defined as

\[
H(\mathcal{X}) \triangleq \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n)
\]

when it exists.

**Definition 2.5**

Again, assume a stochastic process and define the following rate:

\[
H'(\mathcal{X}) \triangleq \lim_{n \to \infty} H(X_n | X_{n-1}, X_{n-2}, \ldots, X_1)
\]

assuming it exists.
Entropy rates or entropy rate

Rates are the same

Theorem 2.6
We have that for stationary stochastic processes
\[
\lim_{n \to \infty} H(X_n | X_{n-1}, X_{n-2}, \ldots, X_1) \equiv H'(X) \equiv H(X) = \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n)
\]
Entropy rates or entropy rate

- Rates are the same

**Theorem 2.6**

*We have that for stationary stochastic processes*

\[
\lim_{n \to \infty} H(X_n | X_{n-1}, X_{n-2}, \ldots, X_1) \triangleq H'(X) \tag{7}
\]

\[
= H(X) \triangleq \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n) \tag{8}
\]
What is entropy of this random walk

- So, the entropy of the random walk is

$$H(X) = \text{(overall edge uncertainty)}$$

- (overall node uncertainty in stationary condition)
What is entropy of this random walk

- So, the entropy of the random walk is

\[ H(X) = (\text{overall edge uncertainty}) \]

\[ \quad - (\text{overall node uncertainty in stationary condition}) \]

- Intuition: As node entropy decreases while keeping edge uncertainty constant, the network becomes more concentrated,
What is entropy of this random walk

- So, the entropy of the random walk is

$$H(X) = (\text{overall edge uncertainty})$$

- (overall node uncertainty in stationary condition)

- Intuition: As node entropy decreases while keeping edge uncertainty constant, the network becomes more concentrated,

- fewer nodes are hubs, and the hubs that remain are widely connected (since edge entropy is fixed).
What is entropy of this random walk

- So, the entropy of the random walk is

\[ H(\mathcal{X}) = (\text{overall edge uncertainty}) \]

- (overall node uncertainty in stationary condition)

- Intuition: As node entropy decreases while keeping edge uncertainty constant, the network becomes more concentrated, fewer nodes are hubs, and the 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 of the walk.
What is entropy of this random walk

- So, the entropy of the random walk is
  
  \[ H(\mathcal{X}) = (\text{overall edge uncertainty}) \]
  
  \[ - (\text{overall node uncertainty in stationary condition}) \]

- Intuition: As node entropy decreases while keeping edge uncertainty constant, the network becomes more concentrated, fewer nodes are hubs, and the 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 of the walk.

- If node entropy goes up with edge entropy fixed, then many nodes are hubs all with relatively low connectivity, so hitting them doesn’t provide much choice \( \Rightarrow \) random walk entropy goes down.
Hidden Markov models (HMMs)

- An HMM is a distribution \( p(X_{1:n}, Y_{1:n}) \) over \( 2n \) random variables that factors in a particular way.
Hidden Markov models (HMMs)

- An HMM is a distribution $p(X_{1:n}, Y_{1:n})$ over $2n$ random variables that factors in a particular way.
- Easiest way to depict all of the factorization properties is to use a graphical model, as in the below, where $n = 5$:

![Graphical Model]

Let $Y_1, Y_2, ..., Y_n$ be a stationary Markov chain. Let $X_1:n$ be a random function of this Markov chain. I.e.,

$$X_i = \begin{cases} \phi_1(Y_i) & \text{with probability } p_1 \\ \phi_2(Y_i) & \text{with probability } p_2 \\ \vdots \\ \phi_m(Y_i) & \text{with probability } p_m \end{cases}$$

$$= \phi_{N_{X}}(X_i)$$

where $N \in \{1, 2, ..., m\}$ itself is a random variable.
Hidden Markov models (HMMs)

- An HMM is a distribution $p(X_{1:n}, Y_{1:n})$ over $2n$ random variables that factors in a particular way.
- Easiest way to depict all of the factorization properties is to use a graphical model, as in the below, where $n = 5$:

Let $Y_1, Y_2, \ldots, Y_n$ be a stationary Markov chain.
Hidden Markov models (HMMs)

- An HMM is a distribution \( p(X_{1:n}, Y_{1:n}) \) over \( 2n \) random variables that factors in a particular way.
- Easiest way to depict all of the factorization properties is to use a graphical model, as in the below, where \( n = 5 \):

\[
\begin{align*}
&
\begin{array}{c}
\phantom{X} \\
\phantom{Y} \\
X_1 \\
X_2 \\
X_3 \\
X_4 \\
X_5 \\
Y_1 \\
Y_2 \\
Y_3 \\
Y_4 \\
Y_5
\end{array}
\end{align*}
\]

- Let \( Y_1, Y_2, \ldots, Y_n \) be a stationary Markov chain.
- Let \( X_{1:n} \) be a random function of this Markov chain. I.e.,

\[
X_i = \begin{cases} 
\phi_1(Y_i) & \text{with probability } p_1 \\
\phi_2(Y_i) & \text{with probability } p_2 \\
\vdots \\
\phi_m(Y_i) & \text{with probability } p_m 
\end{cases} = \phi_N(X_i) \quad (9)
\]

where \( N \in \{1, 2, \ldots, m\} \) itself is a random variable.
Note that the stochastic process \( X_1, X_2 \ldots \) does not form a Markov chain in general. Why?
Note that the stochastic process $X_1, X_2, \ldots$ does not form a Markov chain in general. Why? because it does not satisfy the first order Markov assumption, nor any order Markov assumption in general.
HMMs

- Note that the stochastic process $X_1, X_2, \ldots$ does not form a Markov chain in general. Why? Because it does not satisfy the first order Markov assumption, nor any order Markov assumption in general.

- If $\{Y_i\}_i$ is stationary, then is $\{X_i\}_i$ a stationary stochastic process?
Note that the stochastic process $X_1, X_2 \ldots$ does not form a Markov chain in general. Why? because it does not satisfy the first order Markov assumption, nor any order Markov assumption in general.

If $\{Y_i\}_i$ is stationary, then is $\{X_i\}_i$ a stationary stochastic process? Yes. Possible HW problem, so no more given here.
HMMs

- Note that the stochastic process $X_1, X_2 \ldots$ does not form a Markov chain in general. Why? because it does not satisfy the first order Markov assumption, nor any order Markov assumption in general.

- If $\{Y_i\}_i$ is stationary, then is $\{X_i\}_i$ a stationary stochastic process? Yes. Possible HW problem, so no more given here.

- We can compute the entropy rate of $\{X_i\}_i$, i.e., $H(\mathcal{X}) = \lim_{n \to \infty} H(X_n|X_{n-1}, \ldots, X_1)$ but it is ugly, so instead we compute upper and lower bounds.
HMMs

- Note that the stochastic process $X_1, X_2 \ldots$ does not form a Markov chain in general. Why? Because it does not satisfy the first order Markov assumption, nor any order Markov assumption in general.
- If $\{Y_i\}_i$ is stationary, then is $\{X_i\}_i$ a stationary stochastic process? Yes. Possible HW problem, so no more given here.
- We can compute the entropy rate of $\{X_i\}_i$, i.e.,
  
  \[
  H(X) = \lim_{n \to \infty} H(X_n | X_{n-1}, \ldots, X_1)
  \]
  
  but it is ugly, so instead we compute upper and lower bounds.

- **Upper bound(s):**

  \[
  H(X_n | X_{n-1}, \ldots, X_1) = H(X_{n+1} | X_n, \ldots, X_2) \geq H(X_{n+1} | X_1, \ldots, X_n)
  \]
  
  \[
  \geq H(X_{n+2} | X_{n+1}, \ldots, X_1) \geq \cdots \geq H(\mathcal{X})
  \]
A lower bound is given by $H(X_n|X_{n-1}, \ldots, X_2, Y_1) \leq H(X)$ because

\[
H(X_n|X_{n-1}, \ldots, X_2, Y_1) = H(X_n|X_{n-1}, \ldots, X_2, X_1, Y_1) \tag{12}
\]

\[
= H(X_n|X_{n-1}, \ldots, X_1, Y_1, Y_0, Y_{-1}, \ldots, Y_{-k}) \tag{13}
\]

\[
= H(X_n|X_{n-1}, \ldots, X_1, Y_1, Y_0, Y_{-1}, \ldots, Y_{-k}, X_0, \ldots, X_{-k}) \tag{14}
\]

\[
\leq H(X_n|X_{n-1}, \ldots, X_1, X_0, \ldots, X_{-k})
\]

\[
= H(X_{n+k+1}|X_{n+k}, \ldots, X_1) \tag{15}
\]
A lower bound is given by $H(X_n|X_{n-1}, \ldots, X_2, Y_1) \leq H(\mathcal{X})$ because

\[
H(X_n|X_{n-1}, \ldots, X_2, Y_1) = H(X_n|X_{n-1}, \ldots, X_2, X_1, Y_1) \tag{12}
\]
\[
= H(X_n|X_{n-1}, \ldots, X_1, Y_1, Y_0, Y_{-1}, \ldots, Y_{-k}) \tag{13}
\]
\[
= H(X_n|X_{n-1}, \ldots, X_1, Y_1, Y_0, Y_{-1}, \ldots, Y_{-k}, X_0, \ldots, X_{-k}) \tag{14}
\]
\[
\leq H(X_n|X_{n-1}, \ldots, X_1, X_0, \ldots, X_{-k})
\]
\[
= H(X_{n+k+1}|X_{n+k}, \ldots, X_1) \tag{15}
\]

So summarizing the bounds on the HMM information rates, we have

\[
H(X_n|X_{n-1}, \ldots, X_1, Y_1) \leq H(\mathcal{X}) \leq H(X_n|X_{n-1}, \ldots, X_1) \tag{16}
\]
HMMs

Lemma 3.1 (ever shrinking sandwich)

\[ H(X_n|X_{n-1}, \ldots, X_1) - H(X_n|X_{n-1}, \ldots, X_1, Y_1) \to 0 \quad (17) \]

Proof.

\[
H(X_n|X_{n-1}, \ldots, X_1) - H(X_n|X_{n-1}, \ldots, X_1, Y_1) \\
= I(X_n; Y_1|X_{n-1}, \ldots, X_1) \leq H(Y_1) \quad (18)
\]
Lemma 3.1 (ever shrinking sandwich)

\[ H(X_n|X_{n-1}, \ldots, X_1) - H(X_n|X_{n-1}, \ldots, X_1, Y_1) \rightarrow 0 \quad (17) \]

Proof.

\[ H(X_n|X_{n-1}, \ldots, X_1) - H(X_n|X_{n-1}, \ldots, X_1, Y_1) = I(X_n; Y_1|X_{n-1}, \ldots, X_1) \leq H(Y_1) \quad (18) \]

Also, \( I(Y_1; X_1, \ldots, X_n) \leq H(Y_1) \) for all \( n \).
Lemma 3.1 (ever shrinking sandwich)

\[ H(X_n|X_{n-1},\ldots,X_1) - H(X_n|X_{n-1},\ldots,X_1,Y_1) \rightarrow 0 \]  

Proof.

\[ H(X_n|X_{n-1},\ldots,X_1) - H(X_n|X_{n-1},\ldots,X_1,Y_1) = I(X_n;Y_1|X_{n-1},\ldots,X_1) \leq H(Y_1) \]  

Also, \( I(Y_1;X_1,\ldots,X_n) \leq H(Y_1) \) for all \( n \).

Now,

\[ \lim_{n \rightarrow \infty} I(Y_1;X_1,\ldots,X_n) = \lim_{n \rightarrow \infty} \sum_{i=1}^{n} I(Y_1;X_i|X_{1:i-1}) \]  

\[ = \sum_{i=1}^{\infty} I(Y_1;X_i|X_{1:i-1}) \leq H(Y) < \infty \]
Lemma 3.1 (ever shrinking sandwich)

\[
H(X_n|X_{n-1}, \ldots, X_1) - H(X_n|X_{n-1}, \ldots, X_1, Y_1) \to 0 \quad (17)
\]

Proof.

\[
H(X_n|X_{n-1}, \ldots, X_1) - H(X_n|X_{n-1}, \ldots, X_1, Y_1) = I(X_n; Y_1|X_{n-1}, \ldots, X_1) \leq H(Y_1) \quad (18)
\]

Also, \( I(Y_1; X_1, \ldots, X_n) \leq H(Y_1) \) for all \( n \).

Now,

\[
\lim_{n \to \infty} I(Y_1; X_1, \ldots, X_n) = \lim_{n \to \infty} \sum_{i=1}^{n} I(Y_1; X_i|X_1:i-1) = \sum_{i=1}^{\infty} I(Y_1; X_i|X_1:i-1) \leq H(Y) < \infty \quad (19)
\]

So an infinite sum is constant, must mean the terms \( \to 0 \) as \( n \to \infty \). Thus, each of the terms \( I(Y_1; X_i|X_1:i-1) \to 0 \) as \( n \to \infty \).
HMM rate summary

Summarizing, we have

$$\lim_{n \to \infty} H(X_n|X_{n-1}, \ldots, X_1, Y_1) = H(\mathcal{X}) = \lim_{n \to \infty} H(X_n|X_{n-1}, \ldots, X_1)$$

(21)
New topic: coding, meaning practical lossless coding of information sources governed by a distribution.
Coding

- New topic: coding, meaning practical lossless coding of information sources governed by a distribution.

- Shannon’s source coding theorem said we can code using $R > H(X)$ bits per source symbol if we use long enough block.
New topic: coding, meaning practical lossless coding of information sources governed by a distribution.

Shannon’s source coding theorem said we can code using $R > H(X)$ bits per source symbol if we use long enough block.

These were “block” codes in that we store and/or transfer one block of symbols at a time, but such codes are not always practical.
New topic: coding, meaning practical lossless coding of information sources governed by a distribution.

Shannon’s source coding theorem said we can code using $R > H(X)$ bits per source symbol if we use long enough block.

These were “block” codes in that we store and/or transfer one block of symbols at a time, but such codes are not always practical.

We seek other coding strategies.
New topic: coding, meaning practical lossless coding of information sources governed by a distribution.

Shannon’s source coding theorem said we can code using $R > H(X)$ bits per source symbol if we use long enough block.

These were “block” codes in that we store and/or transfer one block of symbols at a time, but such codes are not always practical.

We seek other coding strategies.

Variable length symbol codes: One symbol at a time is encoded. Variable length (rather than fixed length) code words. Ex: Huffman coding.
New topic: coding, meaning practical lossless coding of information sources governed by a distribution.

Shannon’s source coding theorem said we can code using $R > H(X)$ bits per source symbol if we use long enough block.

These were “block” codes in that we store and/or transfer one block of symbols at a time, but such codes are not always practical.

We seek other coding strategies.

**variable length symbol codes:** One symbol at a time is encoded. Variable length (rather than fixed length) code words. Ex: Huffman coding.

**Stream codes:** codes that operate on data coming in as a stream and decide codeword depending on current symbol and history. Ex: Arithmetic codes, Lempel-Ziv code (the latter of which is Universal since it does not require $p(x)$).
Definition 4.1 (source code)

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

$$C : \mathcal{X} \rightarrow \mathcal{D}^*$$

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 4.2

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

Definition 4.3 (expected length)

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

$$L(C) = \sum_x p(x) \ell(x)$$

(23)
Definition 4.1 (source code)

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

$$C : \mathcal{X} \rightarrow \mathcal{D}^*$$

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 4.2

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

Definition 4.1 (source code)

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

$$C : \mathcal{X} \rightarrow \mathcal{D}^*$$

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 4.2

Let $\mathcal{X} = \{\text{red, blue}\}$. 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 4.3 (expected length)

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

$$L(C) = \sum_x p(x)\ell(x)$$

(23)
Codes

- Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).
Codes

- Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).
- Another code,

\[
\begin{array}{c|c|c}
   x & p(x) & c(x) \\
   \hline
   1 & \frac{1}{2} & 0 \\
   2 & \frac{1}{4} & 10 \\
   3 & \frac{1}{8} & 110 \\
   4 & \frac{1}{8} & 111 \\
\end{array}
\]

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 010110111111110100?

1, 2, 3, 4, 4, 3, 2, 1

With punctuation: 0, 10, 110, 111, 111, 110, 10, 0, so code in some sense is “self punctuating”.
Codes

- Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).
- Another code,

Example 4.4

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{4}$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{8}$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{8}$</td>
<td>111</td>
</tr>
</tbody>
</table>

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 010110111111110100?

1, 2, 3, 4, 4, 3, 2, 1

With punctuation: 0, 10, 110, 111, 111, 110, 10, 0, so code in some sense is "self punctuating"
Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code,

Example 4.4

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{4}$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{8}$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{8}$</td>
<td>111</td>
</tr>
</tbody>
</table>

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 010110111111110100?

1, 2, 3, 4, 4, 3, 2, 1

With punctuation: 0,10,110,111,111,110,10,0, so code in some sense is "self punctuating".
Codes

- Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).
- Another code,

Example 4.4

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$.
We can define the code with a table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{4}$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{1}{8}$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{8}$</td>
<td>111</td>
</tr>
</tbody>
</table>

In this case, $H(\mathcal{X}) = 1.75$. But $L(\mathcal{C}) = E\ell(\mathcal{X}) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 010110111111110100?

1, 2, 3, 4, 4, 3, 2, 1
With punctuation: 0, 10, 110, 111, 111, 110, 10, 0, so code in some sense is “self punctuating.”
Assume $D = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code,

Example 4.4

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1/2$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$1/4$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$1/8$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$1/8$</td>
<td>111</td>
</tr>
</tbody>
</table>

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.
Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code, let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1/4</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>1/8</td>
<td>111</td>
</tr>
</tbody>
</table>

In this case, $H(X) = 1.75$. 

Example 4.4
Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code,

**Example 4.4**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1/2$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$1/4$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$1/8$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$1/8$</td>
<td>111</td>
</tr>
</tbody>
</table>

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.
Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code,

**Example 4.4**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1/2$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$1/4$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$1/8$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$1/8$</td>
<td>111</td>
</tr>
</tbody>
</table>

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 01011011111110100?
Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code,

**Example 4.4**

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1/2$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$1/4$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$1/8$</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>$1/8$</td>
<td>111</td>
</tr>
</tbody>
</table>

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 01011011111110100? 1, 2, 3, 4, 4, 3, 2, 1
Assume $\mathcal{D} = \{0, 1, 2, \ldots, D - 1\}$ in general (but often $D = 2$).

Another code,

**Example 4.4**

Let $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{D} = \{0, 1\}$. We can define the code with a table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$c(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1/4</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>1/8</td>
<td>111</td>
</tr>
</tbody>
</table>

In this case, $H(X) = 1.75$. But $L(C) = E\ell(X) = 1.75$, so this code is pretty good.

Moreover, it is easy to code. What source symbols correspond to the string 0101101111110100? 1, 2, 3, 4, 4, 3, 2, 1

With punctuation: 0,10,110,111,111,110,10,0, so code in some sense is “self punctuating”
Aside: English

- isenglishselfpunctuating
Aside: English

is English self-punctuating?

how long does it take you to read this sentence that is written without any punctuation marks or even end of sentence markers such as a question mark or even sentence spaces.

Nowhere = “now, here” or “no, where”?
Aside: English

- isenglishselfpunctuating

How long does it take you to read this sentence that is written without any punctuation marks or even end of sentence marks such as question marks or even inter-sentence spaces?

- nowhere = "now, here" or "no, where"?
Aside: English

- isenglishselfpunctuating
How long does it take you to read this sentence that is written without any punctuation marks or even end of sentence marks such as a question mark or even inter-sentence spaces?
Aside: English

- isenglishselfpunctuating

How long does it take you to read this sentence that is written without any punctuation marks or even end of sentence marks such as a question mark or even inter-sentence spaces?

- nowhere = “now, here” or “no, where”?
Another code

Here, $\mathcal{X} = \{1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$
Another code

- Here, $\mathcal{X} = \{1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$

- **Code is:**

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(x)$</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
</tr>
<tr>
<td>$C(X)$</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Another code

Here, $\mathcal{X} = \{1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$

Code is:

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(x)$</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
</tr>
<tr>
<td>$C(X)$</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

So $H = 1.58$ but $E\ell(X) = 1.66 > H$ bits.
Another code

Here, $\mathcal{X} = \{1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$

Code is:

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(x)$</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
</tr>
<tr>
<td>$C(X)$</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

So $H = 1.58$ but $E\ell(X) = 1.66 > H$ bits.

Can we easily decode?
Another code

- Here, $\mathcal{X} = \{1, 2, 3\}$ and $\mathcal{D} = \{0, 1\}$
- Code is:

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(x)$</td>
<td>$1/3$</td>
<td>$1/3$</td>
<td>$1/3$</td>
</tr>
<tr>
<td>$C(X)$</td>
<td>0</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

- So $H = 1.58$ but $E\ell(X) = 1.66 > H$ bits.
- Can we easily decode? $10110010 = 2,3,1,1,2$
Ex: Morse Code

- Morse code, series of dots and dashes to represent letters

![Morse Code Chart]

Notes:
- Most frequent letter sent with the shortest code, 1 dot.
- Codewords might be prefixes of each other (e.g., "E" and "F").
- Uses only binary data (single current telegraph, size two "alphabet"); could use more (three, double current telegraph), but this is more susceptible to noise (binary in computer rather than ternary).
Ex: Morse Code

- Morse code, series of dots and dashes to represent letters
- most frequent letter sent with the shortest code, 1 dot
Ex: Morse Code

- Morse code, series of dots and dashes to represent letters
- most frequent letter sent with the shortest code, 1 dot
- Note: codewords might be prefixes of each other (e.g., “E” and “F”).
Morse code, series of dots and dashes to represent letters
most frequent letter sent with the shortest code, 1 dot
Note: codewords might be prefixes of each other (e.g., “E” and “F”).
uses only binary data (single current telegraph, size two “alphabet”), could use more (three, double current telegraph), but this is more susceptible to noise (binary in computer rather than ternary).
For $\mathcal{X} = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

$H(X) = 1.75$  $E\ell(X) = - 1.125$  $1.25$  $1.75$  $1.875$

The one with the shortest expected length is generally preferable, and so would we prefer $C_I$ or $C_{II}$? But how can the expected length be smaller than entropy? Consider $C_I$ and decode string: 00001. It could come from 1,2,1,2,3 or 2,1,2,1,3 or 1,1,1,1,3, or etc. This code is undesirable since we can't decode, or if we try we will have errors with very high probability.
Set of codes

For $\mathcal{X} = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

$H(X) = 1.75$, $E\ell(X) = 1.875$

Code efficiency $= \frac{H(X)}{E\ell(X)}$. 

Which code is best? The one with the shortest expected length is generally preferable, and so would we prefer $C_I$ or $C_{II}$? But how can the expected length be smaller than entropy? Consider $C_I$ and decode string: 00001. It could come from 1,2,1,2,3 or 2,1,2,1,3 or 1,1,1,1,3, or etc. This code is undesirable since we can't decode, or if we try we will have errors with very high probability.
Set of codes

For $\mathcal{X} = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

$H(X) = 1.75$, $E\ell(X) = 1.125, 1.25, 1.75, 1.875$

- Code efficiency = $H(X)/E\ell(X)$.
- Which code is best?
Set of codes

For $\mathcal{X} = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>$H(X)$</td>
<td>1.75</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$E\ell(X)$</td>
<td>-</td>
<td>1.125</td>
<td>1.25</td>
<td>1.75</td>
<td>1.875</td>
</tr>
</tbody>
</table>

Code efficiency $= H(X)/E\ell(X)$.

Which code is best? The one with the shortest expected length is generally preferable, and so would we prefer $C_I$ or $C_{II}$?
Set of codes

- For $\mathcal{X} = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

$H(X) = 1.75$  
$E\ell(X) = 1.125$  

- Code efficiency $= H(X) / E\ell(X)$.

- Which code is best? The one with the shortest expected length is generally preferable, and so would we prefer $C_I$ or $C_{II}$?

- But how can the expected length be smaller than entropy?
For $\mathcal{X} = \{1, 2, 3, 4\}$ and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_1$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

$H(X) = 1.75$  $E\ell(X) = 1.125$  $1.25$  $1.75$  $1.875$

Code efficiency = $H(X)/E\ell(X)$.

Which code is best? The one with the shortest expected length is generally preferable, and so would we prefer $C_1$ or $C_{II}$?

But how can the expected length be smaller than entropy?

Consider $C_1$ and decode string: 00001. It could come from 1,2,1,2,3 or 2,1,2,1,3 or 1,1,1,1,3, or etc.
Set of codes

- For \( X = \{1, 2, 3, 4\} \) and binary code, consider the following 4 codes.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( p(x) )</th>
<th>( C_I )</th>
<th>( C_{II} )</th>
<th>( C_{III} )</th>
<th>( C_{IV} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>( H(X) )</td>
<td>1.75</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( E\ell(X) )</td>
<td>-</td>
<td>1.125</td>
<td>1.25</td>
<td>1.75</td>
<td>1.875</td>
</tr>
</tbody>
</table>

- Code efficiency = \( H(X)/E\ell(X) \).

- Which code is best? The one with the shortest expected length is generally preferable, and so would we prefer \( C_I \) or \( C_{II} \)?
- But how can the expected length be smaller than entropy?
- Consider \( C_I \) and decode string: 00001. It could come from 1,2,1,2,3 or 2,1,2,1,3 or 1,1,1,1,3, or etc. This code is undesirable since we can’t decode, or if we try we will have errors with very high probability.
### Set of codes

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_1$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

- $H(X) = 1.75$  
- $E_\ell(X) = 1.125$  
- $E_\ell(X) = 1.25$  
- $E_\ell(X) = 1.75$  
- $E_\ell(X) = 1.875$

- Consider $C_{II}$. How would we decode 0011?
## Set of codes

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

$H(X)$ = 1.75  
$E\ell(X)$ = - 1.125 | 1.25 | 1.75 | 1.875

- Consider $C_{II}$. How would we decode 0011? **Could be either 1,1,2,2 or 3,4, so again we can’t decode without a probability of error (and the longer the sequence the probability of error goes to 1).**
Consider $C_{II}$. How would we decode 0011? Could be either 1,1,2,2 or 3,4, so again we can’t decode without a probability of error (and the longer the sequence the probability of error goes to 1).

Consider $C_{III}$. This code seems at least feasible (since $E\ell \geq H$). Decoding seems easy: (e.g., 111110100 = 111,110,10,0 = 4,3,2,1) seems like every string we encounter easily gets to the end of a codeword before going to next codeword.
Consider $C_{IV}$. Can we decode 00000000111?
## Set of codes

<table>
<thead>
<tr>
<th>$x$</th>
<th>$p(x)$</th>
<th>$C_I$</th>
<th>$C_{II}$</th>
<th>$C_{III}$</th>
<th>$C_{IV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>$H(X)$</td>
<td>1.75</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$E\ell(X)$</td>
<td>-</td>
<td>1.125</td>
<td>1.25</td>
<td>1.75</td>
<td>1.875</td>
</tr>
</tbody>
</table>

- Consider $C_{IV}$. Can we decode 00000000111? Yes, but if we only see a prefix, such as 01, we don’t know until we see more bits what source symbol it is.
Definition 4.5 (non-singular)

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

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j)$$  \hspace{1cm} (24)

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

Definition 4.5 (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 $D^*$. I.e.,

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j) \quad (24)$$

- 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

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

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).
Our goal

- 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 there 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.
Our goal

- 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.
Our goal

- 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 4.6 (code extension)**

A code extension $C^*$ of $C$ is a mapping from finite length strings of $\mathcal{D}$, defined as:

$$C(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n)$$ (25)
**Our goal**

- 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 there 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 4.6 (code extension)**

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

$$C(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2)\ldots C(x_n) \quad (25)$$

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

- 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 ∃ 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 4.6 (code extension)**

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

$$C(x_1, x_2, \ldots, x_n) = C(x_1)C(x_2) \ldots C(x_n) \quad (25)$$

- 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**

**Definition 4.7 (uniquely decodable)**

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

- $C_1$ singular.
Definition 4.7 (uniquely decodable)

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

- $C_I$ singular. **Extension to $C_{II}$** singular so $C_{II}$ not uniquely decodable.
Code types

Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_{11}$ singular so $C_{11}$ not uniquely decodable.
- **But how long must we wait until we know the source?**
Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_{II}$ singular so $C_{II}$ 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.
Code types

**Definition 4.7 (uniquely decodable)**

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

- $C_1$ singular. Extension to $C_{II}$ singular so $C_{II}$ 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

\[
\begin{align*}
C(1) & = 10 \\
C(2) & = 00 \\
C(3) & = 11 \\
C(4) & = 110 \\
\end{align*}
\]
Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_1\|$ singular so $C_1\|$ 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

$$x_1 x_2 x_3 x_4$$

Code string: $1100000000 = 3,2,2,2,2$

Code string: $11000000000 = 4,2,2,2,2$

So we don’t know identity of first symbol until end of code string.
Code types

**Definition 4.7 (uniquely decodable)**

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

- \( C_I \) singular. Extension to \( C_{II} \) singular so \( C_{II} \) 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

\[
\begin{align*}
C_1 &= 10, 00, 11, 110 \\
C_{II} &= 3, 2, 2, 2, 2 \\
C_{III} &= 4, 2, 2, 2, 2
\end{align*}
\]

So we don't know identity of first symbol until end of code string.
Definition 4.7 (uniquely decodable)

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

- \( C_1 \) singular. Extension to \( C_{11} \) singular so \( C_{11} \) 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

\[
\begin{array}{c|c|c|c|c}
  x & 1 & 2 & 3 & 4 \\
  C(x) & 10 & 00 & 11 & 110 \\
\end{array}
\]
**Code types**

**Definition 4.7 (uniquely decodable)**

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

- $C_1$ singular. Extension to $C_{II}$ singular so $C_{II}$ 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?
Code types

**Definition 4.7 (uniquely decodable)**

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

- $C_1$ singular. Extension to $C_{II}$ singular so $C_{II}$ 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
  
  \[
  \begin{array}{c|cccc}
  x & 1 & 2 & 3 & 4 \\
  C(x) & 10 & 00 & 11 & 110 \\
  \end{array}
  \]

  - Is this code uniquely decodable? Yes.
Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_{11}$ singular so $C_{11}$ 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.
Code types

Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_{11}$ singular so $C_{11}$ 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>

- Code string: 1100000000 =

- Is this code uniquely decodable? Yes.
Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_{II}$ singular so $C_{II}$ 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>

- Code string: 1100000000 = 3,2,2,2,2
- Is this code uniquely decodable? Yes.
Definition 4.7 (uniquely decodable)

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

- $C_1$ singular. Extension to $C_II$ singular so $C_II$ 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>

- Code string: $1100000000 = 3,2,2,2,2$
- Code string: $110000000000 = 4,2,2,2,2,2$

- Is this code uniquely decodable? Yes.
Code types

Definition 4.7 (uniquely decodable)

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

- \( C_1 \) singular. Extension to \( C_{II} \) singular so \( C_{II} \) 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

\[
\begin{array}{c|cccc}
 x & 1 & 2 & 3 & 4 \\
 C(x) & 10 & 00 & 11 & 110 \\
\end{array}
\]

- Code string: 1100000000 = 3,2,2,2,2
- Code string: 11000000000 = 4,2,2,2,2

Is this code uniquely decodable?
Yes.
Code types

Definition 4.7 (uniquely decodable)
A code $C$ with extension $C^*$ is uniquely decodable if the extension $C^*$ is non-singular.

- $C_1$ singular. Extension to $C_{II}$ singular so $C_{II}$ 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>

- Code string: 1100000000 = 3,2,2,2,2
- Code string: 11000000000 = 4,2,2,2,2

- Is this code uniquely decodable? Yes.
- So we don’t know identity of first symbol until end of code string. 😞.
Prefix codes

Definition 4.8 (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.
Prefix codes

Definition 4.8 (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.
Definition 4.8 (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 codes

Definition 4.8 (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.
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).
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 \( x = \text{ungzip} (\text{gzip}(x)) \).
Kraft inequality

**Theorem 4.9 (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*

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

(26)
Kraft inequality

Theorem 4.9 (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

$$\sum_{i} D^{-\ell_i} \leq 1$$  \hspace{1cm} (26)

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

- Note what the converse is saying: there exists a code with these lengths, not that all codes with these lengths will satisfy the inequality.
**Theorem 4.9 (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

$$\sum_i D^{-\ell_i} \leq 1 \quad (26)$$

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

- Note what the converse is saying: 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)
proof of Kraft inequality.

- Represent the set of codes on a $D$-ary tree, as in:
proof of Kraft inequality.

- Represent the set of codes on a $D$-ary tree, as in:
proof of Kraft inequality.

- Represent the set of codes on a $D$-ary tree, as in:

![D-ary tree diagram]

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 the set of codes on a \( D \)-ary tree, as in:

![Diagram of a \( D \)-ary tree representing the set of codes.](image)

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 the set of codes on a \( D \)-ary tree, as in:

  ![](image)

  - Codewords correspond to leaves.
Kraft inequality

proof of Kraft inequality.

- Represent the set of codes on a $D$-ary tree, as in:

- Codewords correspond to leaves
- Path from root to leaf determines a codeword
proof of Kraft inequality.

- Represent the set of codes on a $D$-ary tree, as in:

- 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)
Kraft inequality

...proof of Kraft inequality cont.

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

...
... 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 \(\ell_{\text{max}}\).

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

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

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 \( \leq D \ell_{\text{max}} \).
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,
Kraft inequality

... proof of Kraft inequality cont.

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

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

1. codewords,
2. descendants of codewords, or
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
\( \ell_{\text{max}} = \max_i (\ell_i) \) 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 \( \ell_{\text{max}} \).
  
  ![Tree Diagram]

  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 \( \ell_{\text{max}} \).
\( \ell_{\text{max}} = \max_i(\ell_i) \) 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).
... 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 \( \leq D^{\ell_{\text{max}}} \).
Kraft inequality

... proof of Kraft inequality cont.

- All of the above implies:

\[ \sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \implies \sum_i D^{-\ell_i} \leq 1 \]  

(27)
Kraft inequality

... proof of Kraft inequality cont.

- All of the above implies:

\[
\sum_i D^\ell_{\text{max}} - \ell_i \leq D^\ell_{\text{max}} \Rightarrow \sum_i D^{-\ell_i} \leq 1 \quad (27)
\]

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

- All of the above implies:

$$\sum_i D^{\ell_{\max} - \ell_i} \leq D^{\ell_{\max}} \Rightarrow \sum_i D^{-\ell_i} \leq 1 \quad (27)$$

- 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 $\ell_{\max}$ with $D^{\ell_{\max}}$ terminal nodes.
Kraft inequality

...proof of Kraft inequality cont.

All of the above implies:

$$\sum_i D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \Rightarrow \sum_i D^{-\ell_i} \leq 1 \quad (27)$$

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 $1/D^2$, etc.
Kraft inequality

\[ \sum_{i} D^{\ell_{\text{max}} - \ell_i} \leq D^{\ell_{\text{max}}} \implies \sum_{i} D^{-\ell_i} \leq 1 \quad (27) \]

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.

- At level 0, there is a fraction 1 of the descendants at each node at that level,
- At level 1, there is a fraction $1/D$ descendants at each node at that level,
- At level 2, there is a fraction $1/D^2$, etc.

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 $i$. 

...
Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) 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 \((\ell_1, \ell_2, \ldots, \ell_m)\) 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 code.
Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) 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 code.

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).
Kraft inequality

... proof of Kraft inequality cont.

- Sort the lengths $(\ell_1, \ell_2, \ldots, \ell_m)$ 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 code.

- 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 $s_2$, thus eliminating a fraction $D^{-s_2}$ of the nodes.
Kraft inequality

... proof of Kraft inequality cont.

- Sort the lengths \((\ell_1, \ell_2, \ldots, \ell_m)\) 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 code.

- 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 \(s_2\), thus eliminating a fraction \(D^{-s_2}\) of the nodes.

- Total 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).
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 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 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.
Infinite Kraft

**Theorem 4.10 (countably infinite Kraft)**

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

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

(28)

*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 \( \{0, 1, \ldots, D - 1\} \).
Theorem 4.10 (countably infinite Kraft)

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

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

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 binary fractional digits:

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

\[ (29) \]
Kraft inequality

... proof of infinite Kraft.

- Consider expansion of codeword using binary fractional digits:

\[ 0.y_1y_2y_3 \ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j} \]  \hspace{1cm} (29)

- 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 binary fractional digits:

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

(29)

- 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 \([0.y_1 y_2 \ldots y_{\ell_i} , 0.y_1 y_2 \ldots y_{\ell_i} + 1/D^{\ell_i})\)
Kraft inequality

... proof of infinite Kraft.

- Consider expansion of codeword using binary fractional digits:

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

(29)

- 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 binary fractional digits:

\[ 0.y_1y_2y_3 \ldots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j} \]  \hfill (29)

- 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 $y_1y_2y_3\ldots y_\ell_i$ corresponds to the set of all real numbers that begins with $0.y_1y_2y_3\ldots y_\ell_i$.
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.
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 $D - \ell_i$.
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 $D^{-\ell_i}$. 
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 $D^{-\ell_i}$.
- And since all intervals live in $[0, 1)$ we must have

$$\sum_{i} D^{-\ell_i} \leq 1 \quad (30)$$
...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 $D^{-\ell_i}$.

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

$$\sum_i D^{-\ell_i} \leq 1 \quad (30)$$

- 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 ⇔ Kraft inequality.
Towards Optimal Codes

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

$$L(C) = \sum_{i} p_i \ell_i$$

This is a constrained optimization problem:

$$\min \{ \ell_1 : m \in \mathbb{Z}^+ \}$$

subject to

$$\sum_{i} D - \ell_i \leq 1$$

Linear integer program is an NP-complete optimization, not likely to be efficiently solvable (unless P=NP).
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

$$L(C) = \sum_i p_i \ell_i$$ (31)
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

$$L(C) = \sum_i p_i \ell_i$$ (31)

- This is an constrained optimization problem:

$$\min_{\{\ell_1:m\} \in \mathbb{Z}_{++}} \sum_i p_i \ell_i$$ (32)

subject to $\sum_i D^{-\ell_i} \leq 1$
Towards Optimal Codes

- Summarizing: Prefix code ⇔ 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

\[ L(C) = \sum_i p_i \ell_i \]  

This is an constrained optimization problem:

\[
\min_{\{\ell_1: m\} \in \mathbb{Z}^+} \sum_i p_i \ell_i \quad \text{subject to} \quad \sum_i D^{-\ell_i} \leq 1
\]  

- Linear 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} (33)
Towards Optimal Codes

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

$$J = \sum_i p_i \ell_i + \lambda (\sum_i D^{-\ell_i} - 1)$$  \hspace{1cm} (33)

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0$$  \hspace{1cm} (35)

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

$$\Rightarrow \ell_i^* = -\log_D p_i$$  \hspace{1cm} (37)
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} (33)

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0$$ \hspace{1cm} (34)

$$\Rightarrow D - \ell_i = p_i \lambda \ln D$$ \hspace{1cm} (35)

$$\Rightarrow D - \ell_i = p_i \lambda \ln D$$ \hspace{1cm} (37)
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} (33)

- Taking derivatives and setting to 0,

$$ \frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 $$  \hspace{1cm} (34)

$$ \Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} $$  \hspace{1cm} (35)

$$ \Rightarrow \ell_i^* = -\log_D p_i $$  \hspace{1cm} (37)
Towards Optimal Codes

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

$$J = \sum_i p_i \ell_i + \lambda (\sum_i D^{-\ell_i} - 1)$$  \hspace{1cm} (33)

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0$$  \hspace{1cm} (34)

$$\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D}$$  \hspace{1cm} (35)

$$\frac{\partial J}{\partial \lambda}$$  \hspace{1cm} (37)
Towards Optimal Codes

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

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

(33)

- Taking derivatives and setting to 0,

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

(34)

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

(35)

$$\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1$$

(37)
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} (33)

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0$$  \hspace{1cm} (34)

$$\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D}$$  \hspace{1cm} (35)

$$\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0$$  \hspace{1cm} (37)
Towards Optimal Codes

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

\[ J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right) \quad (33) \]

- Taking derivatives and setting to 0,

\[ \frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 \quad (34) \]

\[ \Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} \quad (35) \]

\[ \frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0 \quad \Rightarrow \quad \lambda = 1/ \ln D \quad (36) \]
Towards Optimal Codes

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

$$J = \sum_i p_i \ell_i + \lambda \left( \sum_i D^{-\ell_i} - 1 \right) \quad (33)$$

- Taking derivatives and setting to 0,

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \ln D = 0 \quad (34)$$

$$\Rightarrow D^{-\ell_i} = \frac{p_i}{\lambda \ln D} \quad (35)$$

$$\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0 \quad \Rightarrow \quad \lambda = 1 / \ln D \quad (36)$$

$$\Rightarrow D^{-\ell_i} = p_i \quad (37)$$
Towards Optimal Codes

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

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

(33)

- Taking derivatives and setting to 0,

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

(34)

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

(35)

\[
\frac{\partial J}{\partial \lambda} = \sum_i D^{-\ell_i} - 1 = 0
\]

\[
\Rightarrow \lambda = \frac{1}{\ln D}
\]

(36)

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

yielding

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

(37)
Towards Optimal Codes

- This implies that:

\[ L^* \]  

(38)
Towards Optimal Codes

- This implies that:

\[ L^* = \sum_{i} p_i \ell_i^* \]  (38)
Towards Optimal Codes

This implies that:

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

(38)

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?
Towards Optimal Codes

This implies that:

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

(38)

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?
Towards Optimal Codes

This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = H(X)/\log D \] (38)
Towards Optimal Codes

- This implies that:

\[
L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = H(X)/\log D \quad (38)
\]

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

This implies that:

\[ L^* = \sum_i p_i \ell_i^* = - \sum_i p_i \log_D p_i = H_D(X) = H(X) / \log D \quad (38) \]

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:

\[ L^* = \sum_i p_i \ell_i^* = -\sum_i p_i \log_D p_i = H_D(X) = H(X)/\log D \quad (38) \]

- 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:

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

(38)

- 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?
Optimal Code Lengths

Theorem 4.11

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

$$L \geq H_D(X)$$  \hspace{1cm} (39)

with equality iff $D^{-\ell_i} = p_i$.

Proof.

next time