Suppose we have an instantaneous code for symbols $a_1, \ldots, a_I$, with probabilities $p_1, \ldots, p_I$ and codeword lengths $l_1, \ldots, l_I$. Under each of the following conditions, we can find a better instantaneous code, i.e. one with smaller expected codeword length:

1. If $p_1 < p_2$ and $l_1 < l_2$: Swap the codewords for $a_1$ and $a_2$.
2. If there is a codeword of the form $xby$, where $x$ and $y$ are strings of zero or more bits, and $b$ is a single bit, but there are no codewords of the form $xb'z$, where $z$ is a string of zero or more bits, and $b' \neq b$: Change all the codewords of the form $xby$ to $xy$. (This improves things if none of the $p_i$ are zero, and never makes things worse.)

The Improvements in Terms of Trees

We can view these improvements in terms of the trees for the codes. Here’s an example:

![Tree Diagram]

- Two codewords have the form 01... but none have the form 00... (ie, there’s only one branch out of the 0 node).
- We can therefore improve the code by deleting the surplus node.

Continuing to Improve the Example

The result is the code shown below:

![Tree Diagram]

- Now we note that $a_6$, with probability 0.30, has a longer codeword than $a_1$, which has probability 0.11. We can improve the code by swapping the codewords for these symbols.
The State After These Improvements

- Here's the code after this improvement:

```
0   00  a_0 p_0 = 0.30
     01  a_2 p_2 = 0.20
  NULL

1   10  a_3 p_3 = 0.14
       110 a_7 p_7 = 0.12
       111 a_8 p_8 = 0.13
```

- In general, after such improvements:
  The most improbable symbol will have the longest codeword and there will be at least one other codeword of this length — its "sibling" in the tree. The second-most improbable symbol will also have a codeword of the longest length.

Reminder: Block Codes for Achieving the Entropy

- Last class we proved that Huffman codes are the optimal single symbol codes (plus a warning: top-down splitting does not work).
- We also proved Shannon's first theorem by showing that if we encode long enough blocks we can get the average per-symbol entropy as close as we want to the entropy of the source.
- Our proof used **lossless codes of variable length** (some blocks had codes longer than other blocks). For ease, we used Shannon-Fano codes, but we could also have used Huffman Codes or any other symbol other code which is guaranteed to get within a constant of the entropy.
- There is another way to compress down to the entropy using long blocks; that is to use **lossy codes of fixed length**.

A Final Rearrangement

- The codewords for the most improbable and second-most improbable symbols must have the same length.
- The most improbable symbol’s codeword also has a “sibling” of the same length.
- We can swap codewords to make this sibling be the codeword for the second-most improbable symbol. For the example, the result is:

```
0   00  a_0 p_0 = 0.30
     01  a_2 p_2 = 0.20
  NULL

1   10  a_3 p_3 = 0.14
       110 a_7 p_7 = 0.12
       111 a_8 p_8 = 0.13
```

Another Way to Compress Down to the Entropy

- We get a similar result by supposing that we will always encode \( N \) symbols into a block of exactly \( NR \) bits (fixed length code). Can we do this in a way that is very likely to be decodable?
- Yes, for large values of \( N \). The Law of Large Numbers (LLN) tells us that the sequence of symbols to encode, \( a_{i_1}, \ldots, a_{i_N} \), is very likely to be a “typical” one, for which

\[
\frac{1}{N} \log_2(1/(p_{i_1} \cdots p_{i_N})) = \frac{1}{N} \sum_{j=1}^{N} \log_2(1/p_{i_j})
\]

is very close to the expectation of \( \log_2(1/p_i) \), which is the entropy, \( H(X) = \sum_i p_i \log_2(1/p_i) \). (See Section 4.3 of MacKay’s book.)
- So if we encode all the sequences in this **typical set** in a way that can be decoded, the code will almost always be uniquely decodable.
How Big is the Typical Set?

- Let’s define “typical” sequences as ones where
  \[(1/N) \log_2(1/(p_1 \cdots p_N)) \leq H(X) + \eta/\sqrt{N}\]
  The probability of any such typical sequence will satisfy
  \[p_1 \cdots p_N \geq 2^{-NH(X) - \eta\sqrt{N}}\]
- We scale the margin allowed above \(H(X)\) as \(1/\sqrt{N}\) since that’s how the standard deviation of an average scales. LLN (Chebychev’s inequality) then tells us that most sequences will satisfy this condition, for some large enough value of \(\eta\).
- The total probability for all such sequences can’t be greater than
  \[2^{NH(X) + \eta\sqrt{N}}\]

Encoding sequences in the Typical Set

- The number of “typical” sequences can’t be greater than
  \[2^{NH(X) + \eta\sqrt{N}}\]
- We will be able to encode these sequences in \(NR\) bits if
  \[NR \geq NH(X) + \eta\sqrt{N}\]. (Using any arbitrary code in which we assign each typical sequence to one of the \(2^{NR}\) codes.)
- If \(R > H(X)\), this will be true if \(N\) is sufficiently large.
- How often will a sequence of length \(N\) fail to be in the typical set?
  To answer this, we need to know how many sequences live in the upper “tail” of the distribution of \((1/N) \log_2(1/(p_1 \cdots p_N))\).
- We can define \(H_\delta(X^N)\) to be average codeword length needed for the typical set to leave out only a fraction \(\delta\) of possible sequences. Formally, it is the logarithm of the minimum number of sequences in the \(N^{th}\) extension of \(X\) whose probabilities sum to at least \(1 - \delta\).

Example

- Example: Consider flipping a coin with \(p_{heads} = 0.1\).
- Here are the plots of \(\delta\) vs. \(H_\delta\).
- For large \(N\), \(H_\delta\) becomes almost independent of \(\delta\).

Another Statement of Shannon’s Theorem

- Let \(X\) be an ensemble with entropy \(H(X) = H\) bits.
- Given \(\epsilon > 0\) and \(0 < \delta < 1\), there exists a positive integer \(N_0\) such that for \(N > N_0\),
  \[\left| \frac{1}{N}H_\delta(X^N) - H \right| < \epsilon.\]
- Both sides of the inequality are interesting. The first part tells us that even if the probability of error \(\delta\) is extremely small, the average number of bits per symbol \(\frac{1}{N}H_\delta(X^N)\) needed to specify a long \(N\)-symbol string with vanishingly small error probability does not have to exceed \(H + \epsilon\) bits. We need to have only a tiny tolerance for error, and the number of bits required drops significantly from \(H_0(X)\) to \((H + \epsilon)\).
Another Statement of Shannon’s Theorem

- Let $X$ be an ensemble with entropy $H(X) = H$ bits.
- Given $\epsilon > 0$ and $0 < \delta < 1$, there exists a positive integer $N_0$ such that for $N > N_0$,
  \[ \left| \frac{1}{N} H_\delta(X^N) - H \right| < \epsilon. \]
- What happens if we are yet more tolerant to compression errors? The second part tells us that even if $\delta$ is very close to 1, so that errors are made most of the time, the average number of bits per symbol needed must still be at least $H - \epsilon$ bits.
- These two extremes tell us that regardless of our specific allowance for error, the number of bits per symbol needed is $H$ bits; no more and no less.

An End and a Beginning

Shannon’s Noiseless Coding Theorem is mathematically satisfying. From a practical point of view, though, we still have two problems:

- How can we compress data to nearly the entropy in practice? The number of possible blocks of size $N$ is $I^N$ — huge when $N$ is large. And $N$ sometimes must be large to get close to the entropy by encoding blocks of size $N$.
  Solution: Instead of symbol codes or block codes, we will introduce a more powerful set of codes called stream codes. The most important example is known as arithmetic coding (coming next).
- Where do the symbol probabilities $p_1, \ldots, p_I$ come from? And are symbols really independent, with known, constant probabilities? This is the problem of source modeling.
  Solution: adaptive methods, which update their estimates of the source model as they encode more and more data. (We’ll see these shortly.)