CSC 310: Information Theory

University of Toronto, Fall 2011

Instructor: Radford M. Neal

Week 3

Tradeoffs Choosing Codeword Lengths

The Kraft-McMillan inequalities imply that to make some codewords shorter, we will have to make others longer.

Example: The obvious binary encoding for eight symbols uses codewords that are all three bits long. This code is instantaneous, and satisfies the Kraft inequality, since:

$$\frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} \quad = \quad 1$$

Suppose we want to encode the first symbol using only two bits. We'll have to make some other codewords longer – eg, we can encode two of the other symbols in four bits, and the remaining five symbols in three bits, since

$$\frac{1}{2^2} + \frac{1}{2^4} + \frac{1}{2^4} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^3} \quad = \quad 1$$

How should we choose among the possible codes?

Formalizing Which Codes are the Best: Probabilities for Source Symbols

We'd like to choose a code that uses short codewords for common symbols and long ones for rare symbols.

To formalize this, we need to assign each symbol in the source alphabet a probability. Let's label the symbols as $a_1, \ldots, a_I$, and write the probabilities as $p_1, \ldots, p_I$. We'll assume that these probabilities don't change with time.

We also assume that symbols in the source sequence, X_1, X_2, $\ldots$, X_N, are *independent*:

$$P(X_1 = a_{i_1},\ X_2 = a_{i_2},\ \ldots,\ X_n = a_{i_N}) \quad = \quad p_{i_1}\, p_{i_2} \cdots p_{i_N}$$

These assumptions are really too restrictive in practice, but we'll ignore that for now.

Expected Codeword Length

Consider a code whose codewords for symbols $a_1, \ldots, a_I$ have lengths $l_1, \ldots, l_I$. Let the probabilities of these symbols be $p_1, \ldots, p_I$. We define the *expected codeword length* for this code to be

$$L_C = \sum_{i=1}^{I} p_i \, l_i$$

This is the average codeword length when encoding a single source symbol. But since averaging is a linear operation, the average length of a coded message with N source symbols is just NL_C. For example, when $N=3$:

$$\begin{aligned} \sum_{i_1=1}^{I} \sum_{i_2=1}^{I} \sum_{i_3=1}^{I} p_{i_1} \, p_{i_2} \, p_{i_3} \, (l_{i_1} + l_{i_2} + l_{i_3}) &= \sum_{i_1=1}^{I} p_{i_1} l_{i_1} + \sum_{i_2=1}^{I} p_{i_2} l_{i_2} + \sum_{i_3=1}^{I} p_{i_3} l_{i_3} \\ &= 3L_C \end{aligned}$$

We aim to choose a code for which L_C is small.

Optimal Codes

We will say that a code is *optimal* for a given source (with given symbol probabilities) if its average length is at least as small as that of any other uniquely-decodable code.

An optimal code always exists; in fact there will be several optimal codes for any source, all with the same average length.

The Kraft-McMillan inequalities imply that one of these optimal codes is *instantaneous*. More generally, for any uniquely decodable code with average length L, there is an instantaneous code that also has average length L.

Questions: Can we figure out the length of an optimal code from the symbol probabilities? Can we find such an optimal code, and use it in practice?

Shannon Information Content

A plausible proposal:

The "amount of information" obtained when we learn that $X = a_i$ is $\log_2(1/p_i)$ bits, where $p_i = P(X = a_i)$. We'll call this the *Shannon information content*.

Example:

We learn which of 64 equally-likely possibilities has occurred. The Shannon information content is $\log_2(64) = 6$ bits. This makes sense, since we could encode the result using codewords that are all 6 bits long, and we have no reason to favour one symbol over another by using a code of varying length.

The real test of whether this is a good measure of information content will be whether it actually helps in developing a good theory.

The Entropy of a Source

The Shannon information content pertains to a single value of the random variable X. We'll measure how much information the value of X conveys *on average* by the expected value of the Shannon information content.

This is called the *entropy* of the random variable (or source), and is symbolized by H:

$$H(X) \quad = \quad \sum_{i=1}^{I} p_i \log_2(1/p_i)$$

where $p_i \; = \; P(X = a_i)$.

When the logarithm is to base 2, as above, the entropy has units of bits. We could use a different base if we wish — when base e is used, the units are called "nats" — but we won't in this course.

Information, Entropy, and Codes

How does this relate to data compression?

A vague idea: Since receipt of symbol a_i conveys $\log_2(1/p_i)$ bits of "information", this symbol "ought" to be encoded using a codeword with that many bits. Problem: $\log_2(1/p_i)$ isn't always an integer.

A consequence: If this is done, then the expected codeword length will be equal to the entropy: $\sum_i p_i \log_2(1/p_i)$.

A vague conjecture: The expected codeword length for an optimal code ought to be equal to the entropy.

But this can't quite be right as stated, because $\log_2(1/p_i)$ may not be an integer. Consider $p_0 = 0.1$, $p_1 = 0.9$, so $H = 0.469$. But the optimal code for a symbol with only two values obviously uses codewords 0 and 1, with expected length of 1.

A Property of the Entropy

For any two probability distributions, $p_1, \ldots, p_I$ and $q_1, \ldots, q_I$:

$$\sum_{i=1}^{I} p_i \log_2 \left(\frac{1}{p_i}\right) \;\leq\; \sum_{i=1}^{I} p_i \log_2 \left(\frac{1}{q_i}\right)$$

Proof:

First, note that for all $x > 0$, $\ln x \leq x - 1$. So $\log_2 x \leq (x-1)/\ln 2$.

Now look at the LHS $-$ RHS above:

$$\sum_{i=1}^{I} p_i \left[\log_2 \left(\frac{1}{p_i}\right) - \log_2 \left(\frac{1}{q_i}\right)\right] \;=\; \sum_{i=1}^{I} p_i \log_2 \left(\frac{q_i}{p_i}\right)$$

$$\leq \;\frac{1}{\ln 2} \sum_{i=1}^{I} p_i \left(\frac{q_i}{p_i} - 1\right) \;=\; \frac{1}{\ln 2} \left(\sum_{i=1}^{I} q_i - \sum_{i=1}^{I} p_i\right) \;=\; 0$$

Proving We Can't Compress to Less Than the Entropy

We can use this result to prove that any uniquely decodable binary code for X must have expected length of at least $H(X)$:

Proof: Let the codeword lengths be $l_1, \ldots, l_I$, and let $K = \sum_{i=1}^{I} 2^{-l_i}$ and $q_i = 2^{-l_i}/K$.

The q_i can be seen as probabilities, so

$$\begin{aligned} H(X) &= \sum_{i=1}^{I} p_i \log_2 \left(\frac{1}{p_i}\right) \leq \sum_{i=1}^{I} p_i \log_2 \left(\frac{1}{q_i}\right) \\ &= \sum_{i=1}^{I} p_i \log_2 (2^{l_i} K) = \sum_{i=1}^{I} p_i (l_i + \log_2 K) \end{aligned}$$

Since the code is uniquely decodable, $K \leq 1$ and hence $\log_2 K \leq 0$.
From this, we can conclude that $L_C = \sum p_i l_i \geq H(X)$.

Shannon-Fano Codes

Though we can't always choose codewords with the "right" lengths, $l_i = \log_2(1/p_i)$, we can try to get close. *Shannon-Fano codes* are constructed so that the codewords for the symbols, with probabilities $p_1, \ldots, p_I$, have lengths

$$l_i = \lceil \log_2(1/p_i) \rceil$$

Here, $\lceil x \rceil$ is the smallest integer greater than or equal to x.

The Kraft inequality says such a code exists, since

$$\sum_{i=1}^{I} \frac{1}{2^{l_i}} \leq \sum_{i=1}^{I} \frac{1}{2^{\log_2(1/p_i)}} = \sum_{i=1}^{I} p_i = 1$$

Example:

p_i:	0.4	0.3	0.2	0.1
$\log_2(1/p_i)$:	1.32	1.74	2.32	3.32
$l_i = \lceil \log_2(1/p_i) \rceil$:	2	2	3	4
Codeword:	00	01	100	1100

Expected Lengths of Shannon-Fano Codes

The expected length of a Shannon-Fano code for a souce with symbol probabilities $p_1, \ldots, p_I$, is

$$\begin{aligned}
\sum_{i=1}^{I} p_i l_i &= \sum_{i=1}^{I} p_i \lceil \log_2(1/p_i) \rceil \\
&< \sum_{i=1}^{I} p_i \left(1 + \log_2(1/p_i)\right) \\
&= \sum_{i=1}^{I} p_i \; + \; \sum_{i=1}^{I} p_i \log_2(1/p_i)) \\
&= 1 + H(X)
\end{aligned}$$

This gives an *upper bound* on the expected length of an optimal code. However, the Shannon-Fano code itself may not be optimal (though it sometimes is).

What Have We Shown?

We've now proved the following:

> For a source X, any optimal instantaneous code, C, will have expected length, L_C, satisfying
>
> $$H(X) \leq L_C < H(X) + 1$$

Two main theoretical problems remain:

- Can we find such an optimal code in practice?
- Can we somehow close the gap between $H(X)$ and $H(X) + 1$ above, to show that the entropy is the exactly correct way of measuring the average information content of a source?

Ways to Improve an Instantaneous Code

Suppose we have an instantaneous code for symbols $a_1, \ldots, a_I$, with probabilities $p_1, \ldots, p_I$. Let l_i be the length of the codeword for a_i.

Under each of the following conditions, we can find a better instantaneous code, with smaller expected codeword length:

- If $p_1 < p_2$ and $l_1 < l_2$:

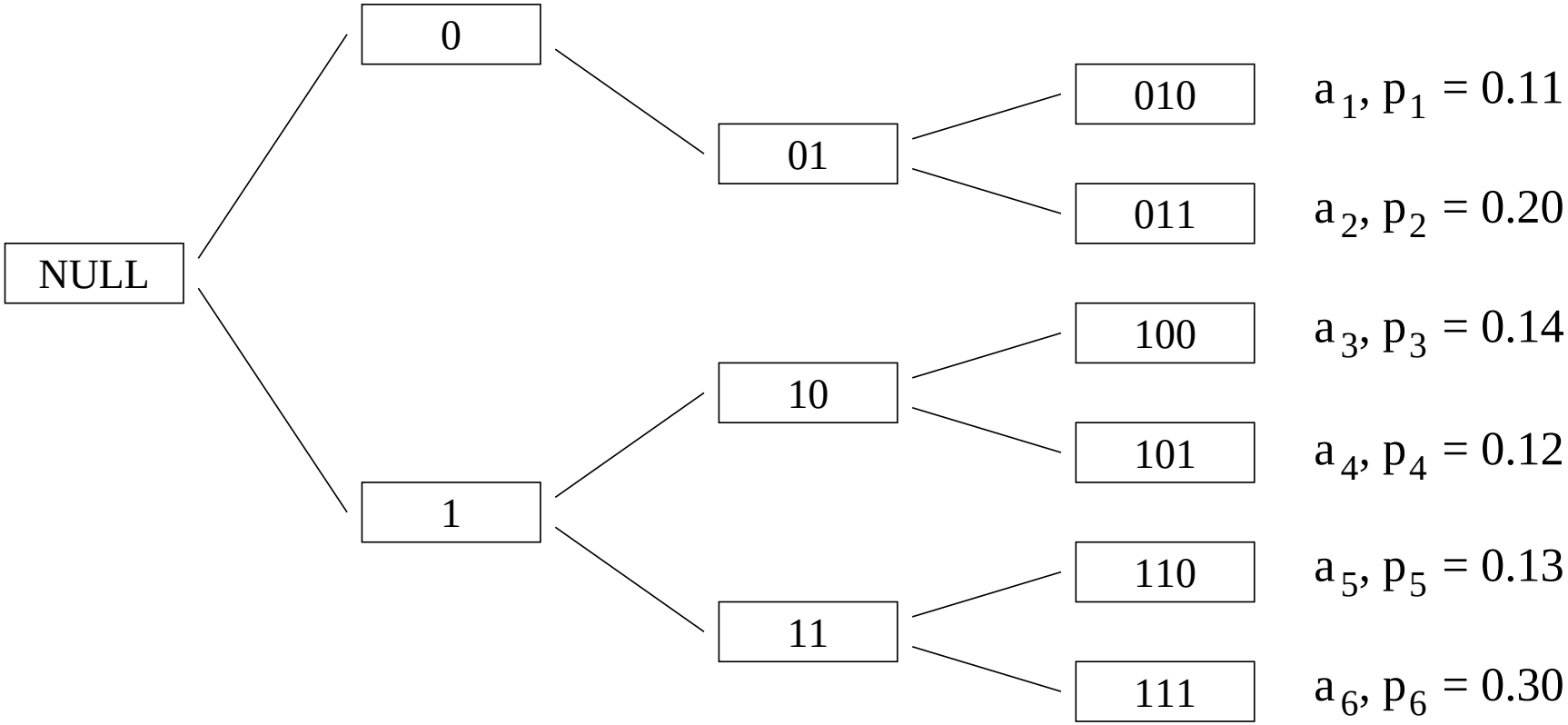
  Swap the codewords for a_1 and a_2.

- 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. (Improves things if none of the p_i are zero, and never makes things worse.)

The Improvements in Terms of Trees

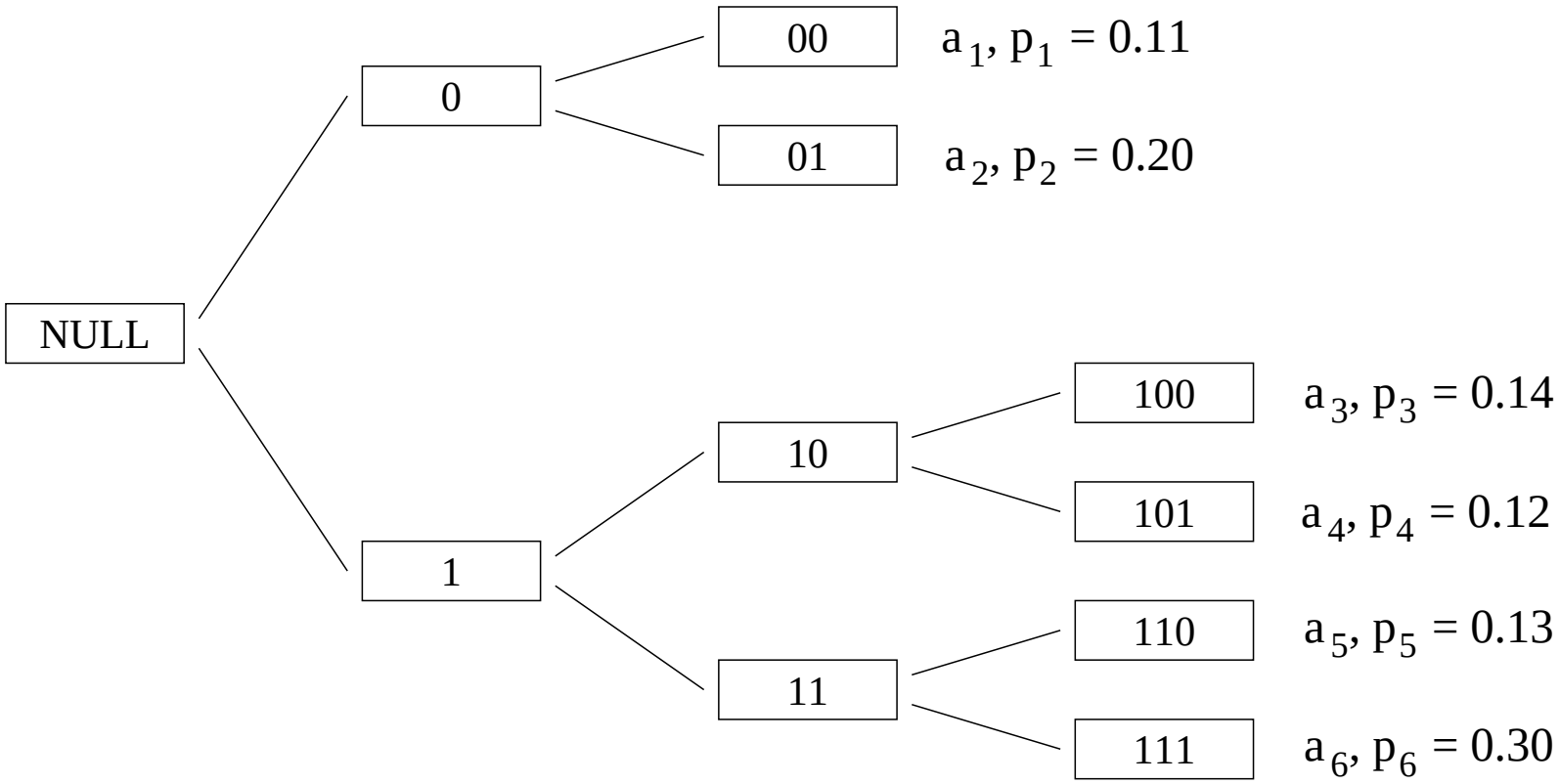
We can view these improvements in terms of the trees for the codes. Here's an example:



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

Continuing to Improve the Example

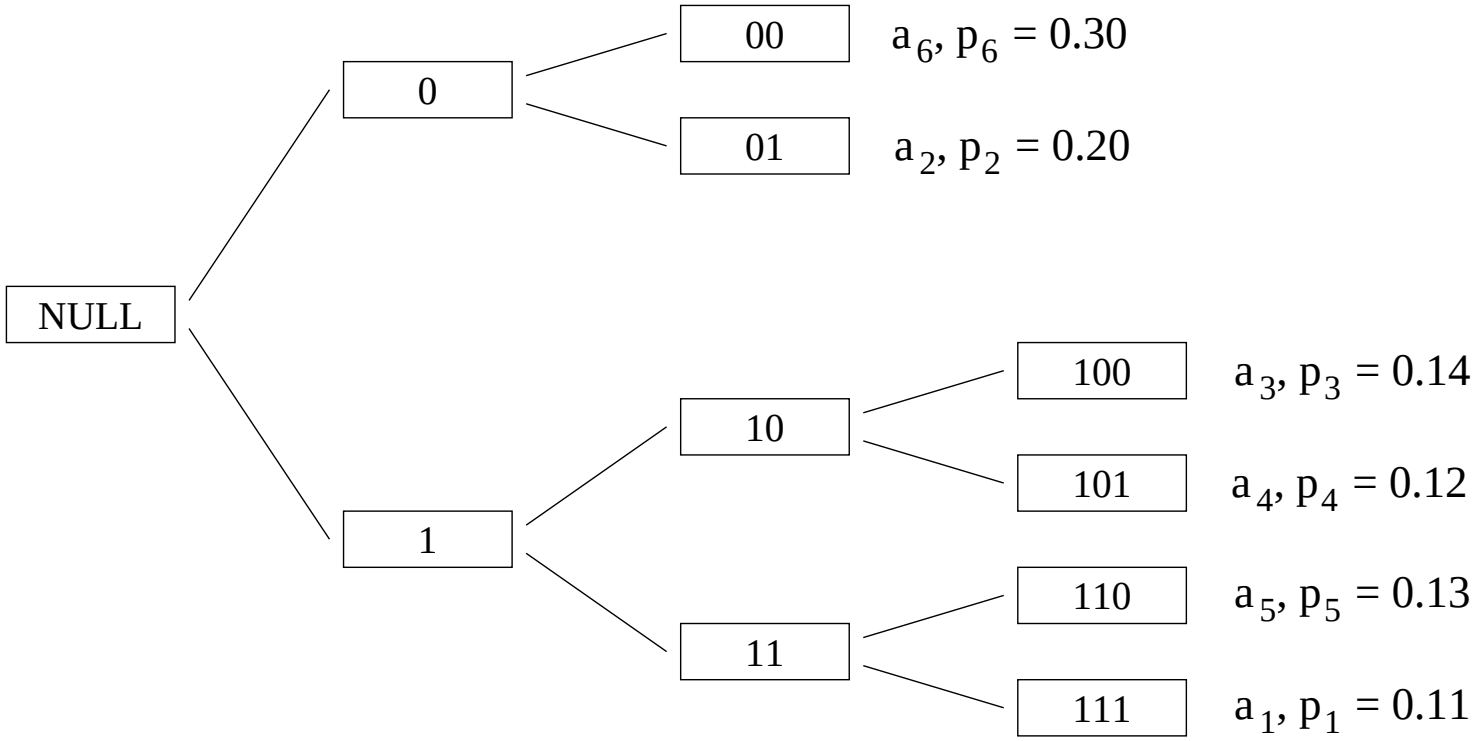
The result is the code shown below:



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 Code After These Improvements

Here's the code after this improvement:



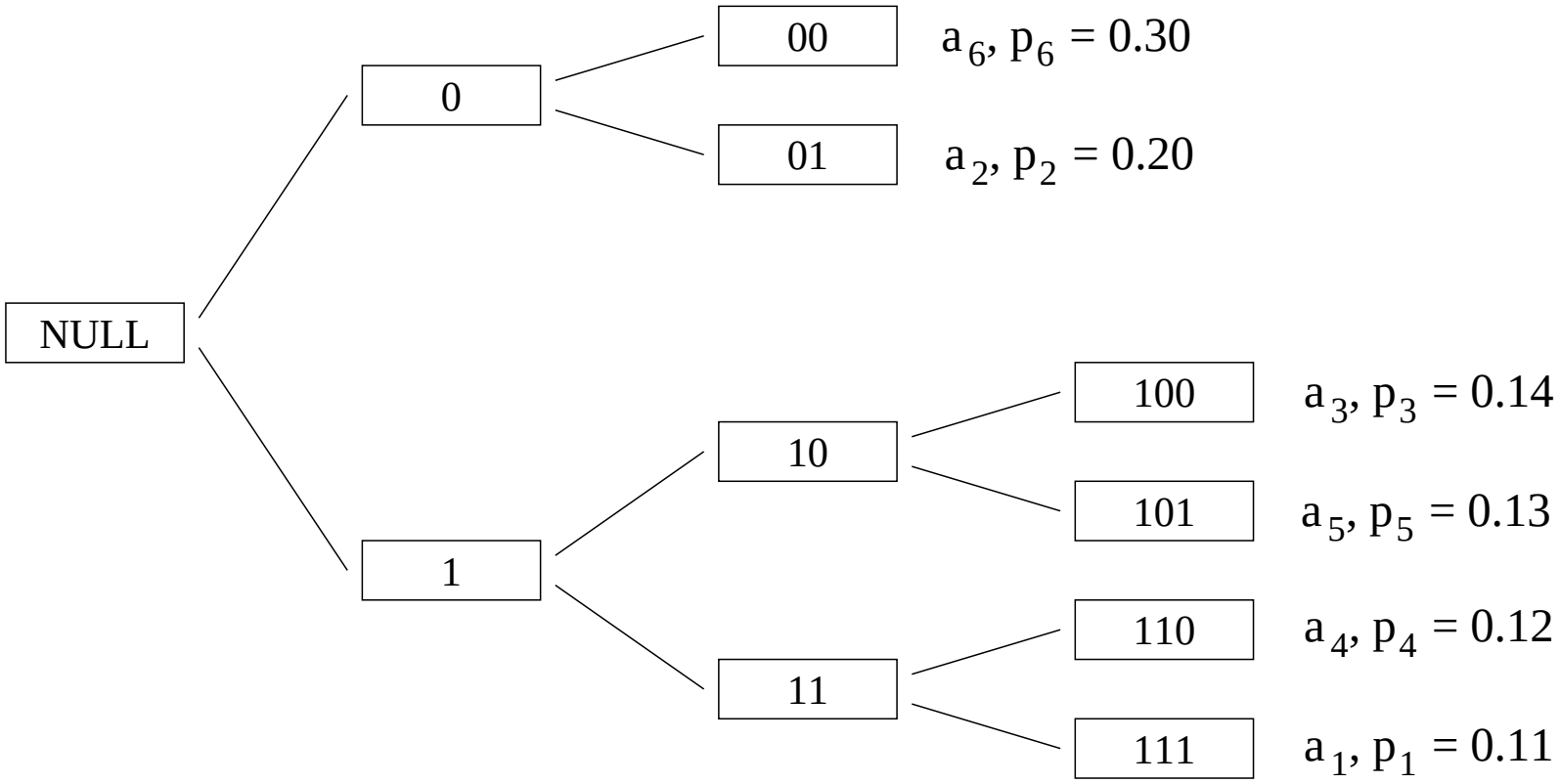
In general, after such improvements:

- One of the least probable symbols will have a codeword of the longest length.
- There will be at least one other codeword of this length — otherwise the longest codeword would be a solitary branch.
- One of the second-most improbable symbols will also have a codeword of the longest length.

A Final Rearrangement

The codewords for the most improbable and second-most improbable symbols must have the same length. Also, the most improbable symbol's codeword must have a "sibling" of the same length.

We can therefore swap codewords to make this sibling be the codeword for the second-most improbable symbol. For the example, the result is



Huffman Codes

We can use these insights about improving code trees to try to construct optimal codes.

We will prove later that the resulting *Huffman codes* are in fact optimal.

We'll concentrate on Huffman codes for a binary code alphabet.
Non-binary Huffman codes are similar, but slightly messier.

Huffman Procedure for Binary Codes

Here's a recursive procedure to construct a binary Huffman code:

procedure Huffman:

inputs: symbols $a_1, \ldots, a_I$, probabilities $p_1, \ldots, p_I$

output: a code mapping $a_1, \ldots, a_I$ to codewords

if $I = 2$:

Return the code $a_1 \mapsto 0,\ a_2 \mapsto 1$.

else

Let $j_1, \ldots, j_I$ be some permutation of $1, \ldots, I$ for which $p_{j_1} \geq \cdots \geq p_{j_I}$.

Create a new symbol a', with associated probability $p' = p_{j_{I-1}} + p_{j_I}$.

Recursively call Huffman to find a code for $a_{j_1}, \ldots, a_{j_{I-2}}, a'$ with probabilities $p_{j_1}, \ldots, p_{j_{I-2}}, p'$. Denote the codewords for $a_{j_1}, \ldots, a_{j_{I-2}}, a'$ in this code by $w_1, \ldots, w_{I-2}, w'$.

Return the code $a_{j_1} \mapsto w_1,\ \ldots,\ a_{j_{I-2}} \mapsto w_{I-2},\ a_{j_{I-1}} \mapsto w'0,\ a_{j_I} \mapsto w'1$.

Proving that Binary Huffman Codes are Optimal

We can prove that the binary Huffman code procedure produces optimal codes by induction on the number of symbols, I.

For $I = 2$, the code produced is obviously optimal — you can't do better than using one bit to code each symbol.

For $I > 2$, we assume that the procedure produces optimal codes for any alphabet of size $I - 1$ (with any symbol probabilities), and then prove that it does so for alphabets of size I as well.

The Induction Step

Suppose the Huffman procedure produces optimal codes for alphabets of size $I - 1$.

Let L be the expected codeword length of the code produced by the Huffman procedure when it is used to encode the symbols $a_1, \ldots, a_I$, having probabilities $p_1, \ldots, p_I$. Without loss of generality, let's assume that $p_i \geq p_{I-1} \geq p_I$ for all $i \in \{1, \ldots, I-2\}$.

The recursive call in the procedure will have produced a code for symbols $a_1, \ldots, a_{I-2}, a'$, having probabilities $p_1, \ldots, p_{I-2}, p'$, with $p' = p_{I-1} + p_I$. By the induction hypothesis, this code is optimal. Let its average length be L'.

The Induction Step (Continued)

Suppose some other instantaneous code for $a_1, \ldots, a_I$ had expected length less than L. We can modify this code so that the codewords for a_{I-1} and a_I are "siblings" (ie, they have the forms $x0$ and $x1$) while keeping its average length the same, or less.

Let the average length of this modified code be $\widehat{L}$, which must also be less than L.

From this modified code, we can make another code for $a_1, \ldots, a_{I-2}, a'$. We keep the codewords for $a_1, \ldots, a_{I-2}$ the same, and encode a' as x. Let the average length of this code be $\widehat{L}'$.

The Induction Step (Conclusion)

We now have two codes for $a_1, \ldots, a_I$ and two for $a_1, \ldots, a_{I-2}, a'$. The average lengths of these codes satisfy the following equations:

$$\begin{aligned} L &= L' + p_{I-1} + p_I \\ \widehat{L} &= \widehat{L}' + p_{I-1} + p_I \end{aligned}$$

Why? The codes for $a_1, \ldots, a_I$ are like the codes for $a_1, \ldots, a_{I-2}, a'$, except that one symbol is replaced by two, whose codewords are one bit longer. This one additional bit is added with probability $p' = p_{I-1} + p_I$.

Since L' is the optimal average length, $L' \leq \widehat{L}'$. From these equations, we then see that $L \leq \widehat{L}$, which contradicts the supposition that $\widehat{L} < L$.

The Huffman procedure therefore produces optimal codes for alphabets of size I. By induction, this is true for all I.

What Have We Accomplished?

We seem to have solved the main practical problem: We now know how to construct an optimal code for any source.

But: This code is optimal **only** if the assumptions we made in formalizing the problem match the real situation.

Often they don't:

- Symbol probabilities may vary over time.
- Symbols may not be independent.
- There is usually no reason to require that X_1, X_2, X_3, ... be encoded one symbol at a time, as $c(X_1)c(X_2)c(X_3)\cdots$.

We would require the last if we really needed instantaneous decoding, but usually we don't.

Example: Black-and-White Images

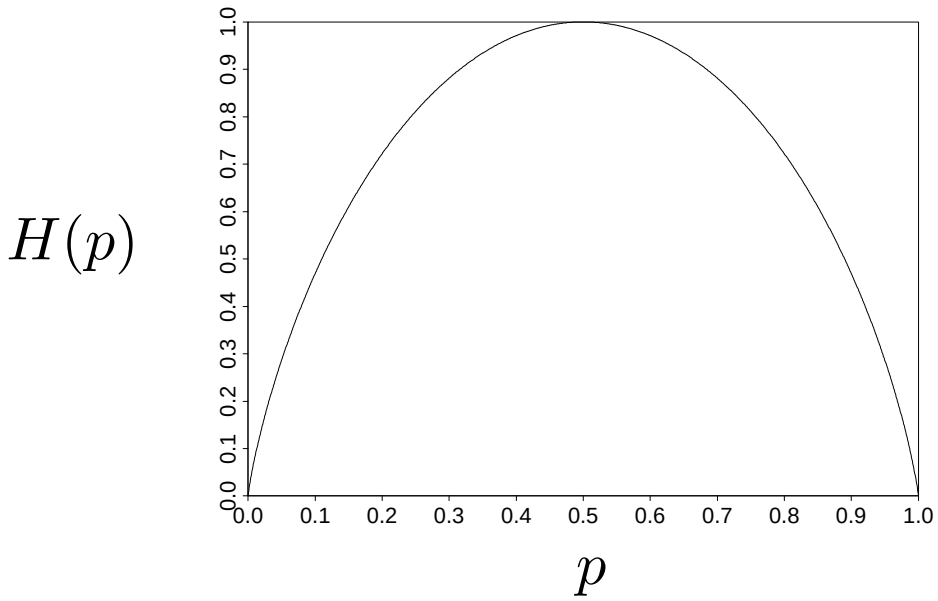
Recall the example from the first lecture, of black-and-white images. There are only two symbols — "white" and "black". The Huffman code is white $\mapsto 0$, black $\mapsto 1$.

This is just the obvious code. But we saw that various schemes such as run length coding can do better than this.

Partly, this is because the pixels are not independent. Even if they were independent, however, we would expect to be able to compress the image if black pixels are much less common than white pixels.

Entropy of a Binary Source

For a binary source, with symbol probabilities p and $1 - p$, the entropy as a function of p looks like this:



For example, $H(0.1) = 0.469$, so we might hope to compress a binary source with symbol probabilities of 0.1 and 0.9 by more than a factor of two. We obviously can't do that if we encode symbols one at a time.

Solution: Coding Blocks of Symbols

We can do better by using Huffman codes to encode *blocks* of symbols.

Suppose our source probabilities are 0.7 for white and 0.3 for black. Assuming pixels are independent, the probabilities for blocks of two pixels will be

$$\begin{array}{ll} \text{white white} & 0.7 \times 0.7 = 0.49 \\ \text{white black} & 0.7 \times 0.3 = 0.21 \\ \text{black white} & 0.3 \times 0.7 = 0.21 \\ \text{black black} & 0.3 \times 0.3 = 0.09 \end{array}$$

Here's a Huffman code for these blocks:

$$WW \mapsto 0,\ WB \mapsto 10,\ BW \mapsto 110,\ BB \mapsto 111$$

The average length for this code is 1.81, which is less than the two bits needed to encode a block of two pixels in the obvious way.