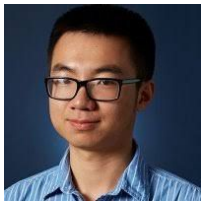
CS294-158 Deep Unsupervised Learning

Lecture 2b: Lossless Compression

Pieter Abbeel, Peter Chen, Jonathan Ho, Aravind Srinivas

UC Berkeley

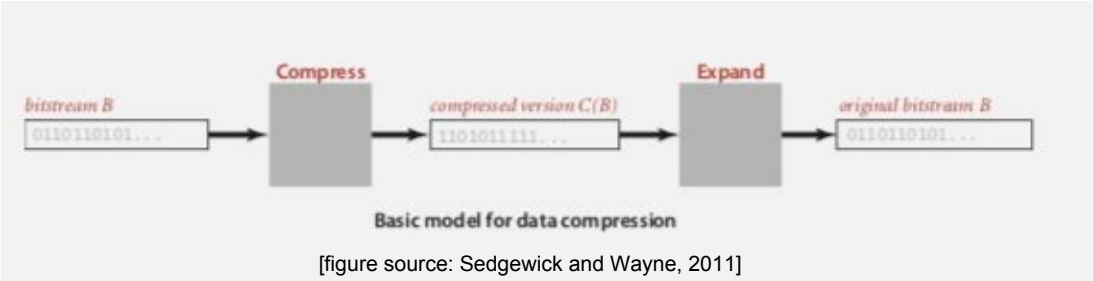
Reading Material

Core reading:

- Introduction to Data Compression, Guy E. Blelloch
  https://www.cs.cmu.edu/~guyb/realworld/compression.pdf

Compression: What and Why?

- What?
  - Reduce number of bits for encoding a message (e.g. image, speech, music, etc…)



[figure source: Sedgewick and Wayne, 2011]

- Why?
  - Save time/bandwidth when transmitting
  - Save space when storing

- Lossy vs. lossless compression [this lecture: lossless]

Applications

- Generic file compression
  - Files: gzip, bzip, 7z
  - Archivers: PKZIP
  - File systems: NTFS, HFS+, ZFS
- Multimedia
  - GIF, JPEG
  - MP3
  - MPEG, DivX, HDTV
- Communication
  - Fax
  - Modem
  - Skype


pied piper
PIED PIPER
WELCOME
TECHNOLOGY
THE CREW
CONTACT

Piper Pied

Original: 327561bytes

New: 163741bytes, 50% reduction

https://techcrunch.com/2015/05/03/ppiper-pied-imitates-hbos-silicon-valley-and-creates-lossless-compression-for-online-images/ (lossy)

Universal Data Compression?

Proposition: No algorithm can compress every bitstring.

Proof [by contradiction]

- Suppose you have a universal data compression algorithm *U* that can compress every bitstream.
- Given bitstring *B*0, compress it to get smaller bitstring *B*1.
- Compress *B*1 to get a smaller bitstring *B*2.
- Continue until reaching bitstring of size 0.
- Implication: all bitstrings can be compressed to 0 bits!

Universal Data Compression?

Proposition: No algorithm can compress every bitstring.

Proof [by counting]

- Suppose your algorithm can compress all 1,000-bit strings.
- 2^{1000} possible bitstrings with 1,000 bits.
- Only $1 + 2 + 4 + ... + 2^{998} + 2^{999}$ can be encoded with ≤ 999 bits.
- Similarly, only 1 in 2^{499} bitstrings can be encoded with ≤ 500 bits!

Why compression possible?

Statistical patterns in the data

"... randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to denmtrasote. In a pubiltacion of New Scnieitst you could ramdinose all the letetrs, keipeng the first two and last two the same, and reibadailty would hadrly be aftcfeed. My ansaylis did not come to much beucase the thoery at the time was for shape and senqeuce retigcionon. Saberi's work sugsegts we may have some pofrweul palrlael prsooscers at work. The resaon for this is suerly that idnetiyfing coentnt by paarllel prseocsing speeds up regnicoiton. We only need the first and last two letetrs to spot chganes in meniang."

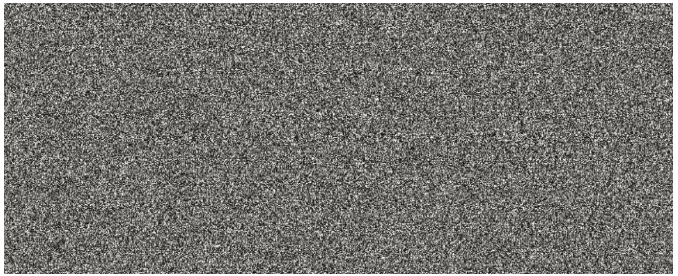
--- Graham Rawlinson, PhD Thesis 1976, Nottingham U.

Why compression possible?

Data with statistical regularities

-> compressible

Random data



-> not compressible

Outline

- Compression: What and Why
- Universal lossless compressor?
- ***Coding of symbols***
  - ***Fixed length***
  - ***Variable length***
  - ***Huffman***
- Theoretical limits
  - Shannon
  - Kraft-McMillan
  - Huffman codes
- Coding Considerations
  - What happens if model \hat{p} only approximation of p?
  - Higher order models
  - +1
- Arithmetic coding
- Lempel-Ziv

(Naive) Coding: Fixed Length

E.g. ASCII: 7 bits per character

USASCII code chart

Bits b_4	b_3	b_2	b_1	Row ↓ / Column → 0	0	1	2	3	4	5	6	7
					b_7 b_6 b_5 = 0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

easy for coding/decoding

but: can't exploit statistical patterns (i.e. doesn't compress)

Variable-length Codes

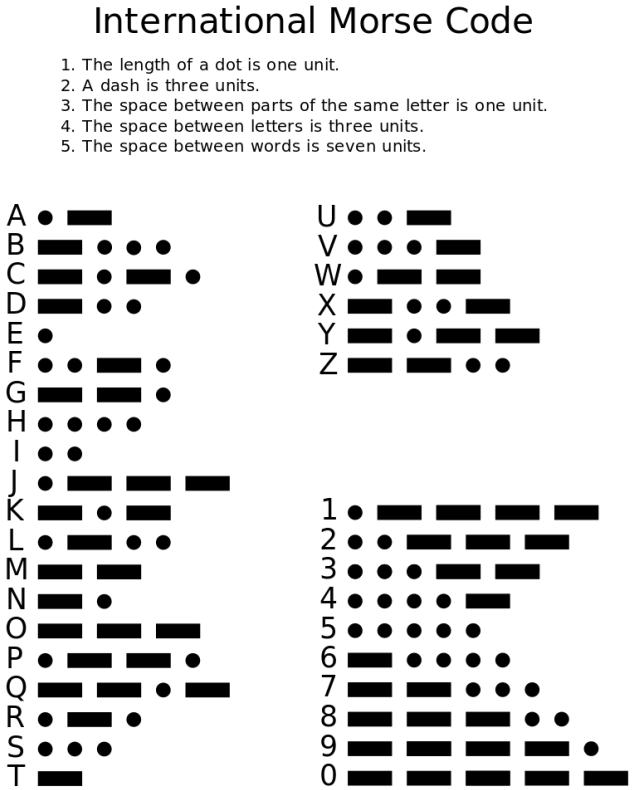
Q: How do we avoid ambiguity?

A: Ensure that no codeword is a prefix of another [sufficient, but not necessary]

- Ex 1. Append special stop char to each codeword (e.g. Morse, but might be wasteful..)
- Ex 2. General prefix-free code

Morse

- Uniquely decodable
- Not prefix-free
- Attaches medium gap to separate codewords



Prefix-free Codes ~~ Binary Tries

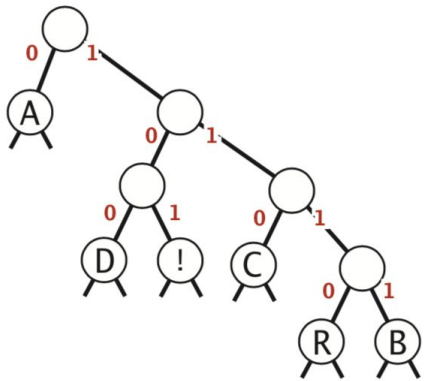
Symbols in leaves

Codeword is path from root to leaf

Codeword table

key	value
!	101
A	0
B	1111
C	110
D	100
R	1110

Trie representation



Compressed bitstring

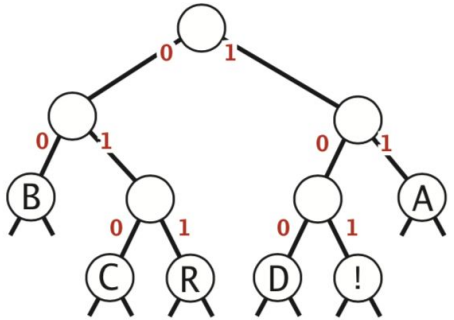
0111111100110010001111111100101 ← 30 bits

A B RA CA DA B RA !

Codeword table

key	value
!	101
A	11
B	00
C	010
D	100
R	011

Trie representation



Compressed bitstring

11000111101011100110001111101 ← 29 bits

A B R A C A D A B R A !

[figure source: Sedgewick and Wayne, 2011]

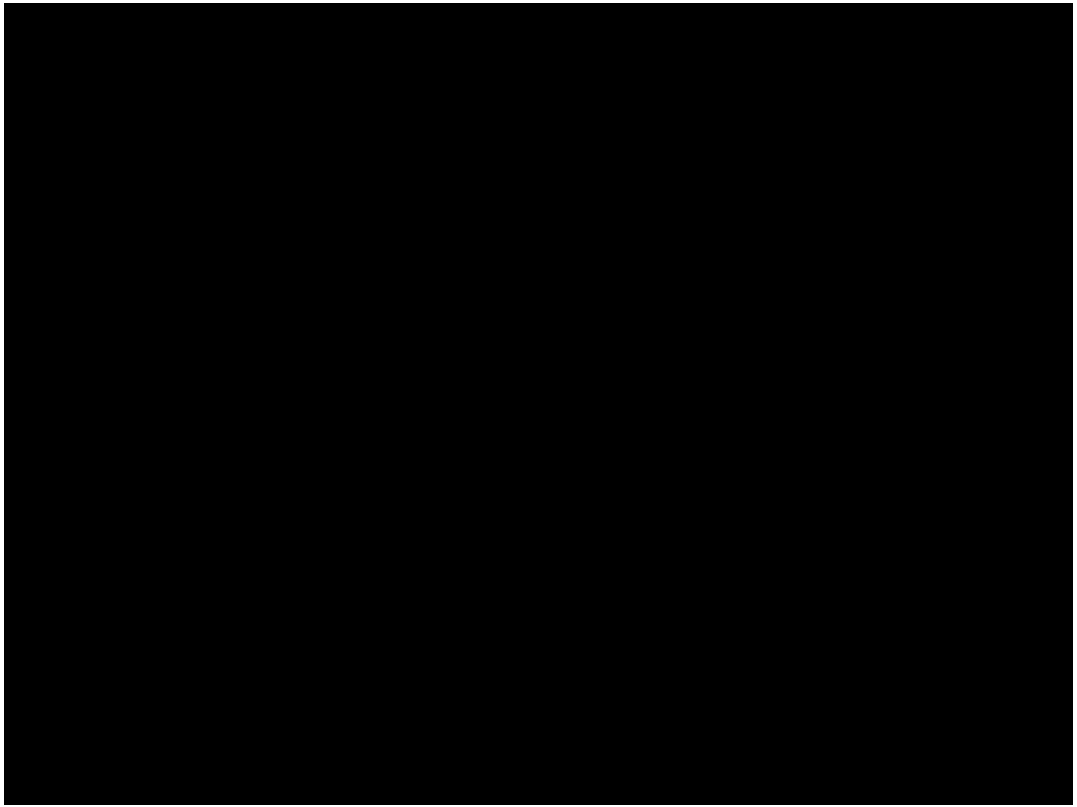
Q: Optimal Prefix-free Code? A:Huffman Codes

Huffman Algorithm:

- Consider probability p[i] of each symbol i in the input
- Start with one node corresponding to each symbol i (with weight p[i])
- Repeat until single trie formed:
  - Select two tries with min probabilities p[k] and p[l]
  - Merge into single trie with probability p[k]+p[l]

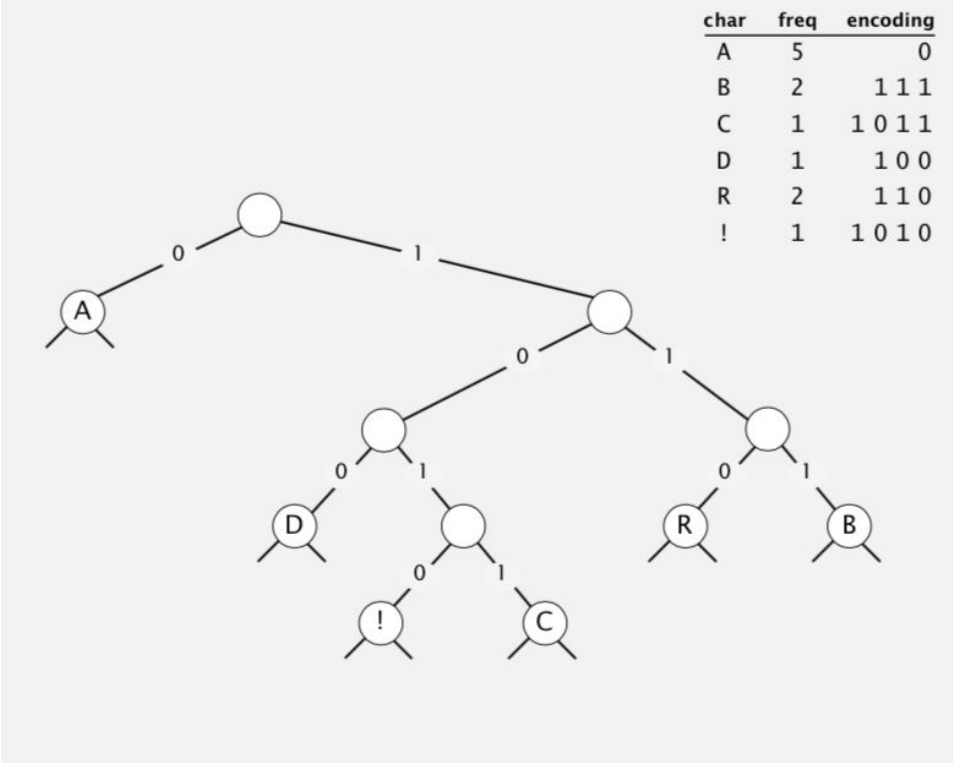
[Why optimal? See later in lecture]

Huffman Codes: Demo



[video source: Sedgewick and Wayne, 2011]

Huffman Codes -- Example



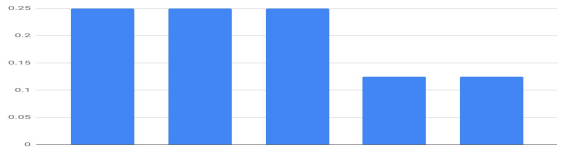
Huffman Code -- Applications

Outline

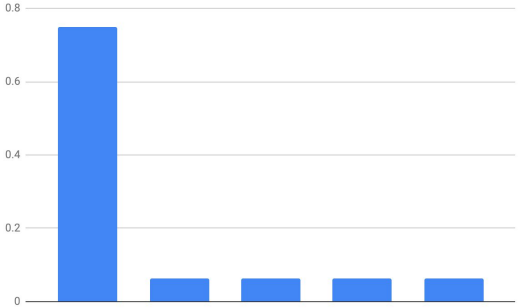
- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
  - Fixed length
  - Variable length
  - Huffman
- ***Theoretical limits***
  - ***Shannon Entropy***
  - ***Kraft-McMillan***
  - ***Optimality of Huffman codes***
- Coding Considerations
  - What happens if model \hat{p} only approximation of p?
  - Higher order models
  - +1
- Arithmetic coding
- Lempel-Ziv

Shannon: Entropy = measure of information

$$H(X) = \sum_i p(x_i) \log_2 \frac{1}{p(x_i)}$$



$$\begin{aligned} p(S) &= \{0.25, 0.25, 0.25, 0.125, 0.125\} \\ H &= 3 \times 0.25 \times \log_2 4 + 2 \times 0.125 \times \log_2 8 \\ &= 1.5 + 0.75 \\ &= 2.25 \end{aligned}$$



$$\begin{aligned} p(s) &= \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\} \\ H &= 0.75 \times \log_2(\frac{4}{3}) + 4 \times 0.0625 \times \log_2 16 \\ &= 0.3 + 1 \\ &= 1.3 \end{aligned}$$

Kraft-McMillan Inequality

- Part I: For any uniquely decodable code C, we have that:

$$\sum_{(s,w)\in C} 2^{-l(w)} \leq 1$$

where l(w) is the length of codeword w for symbol s.

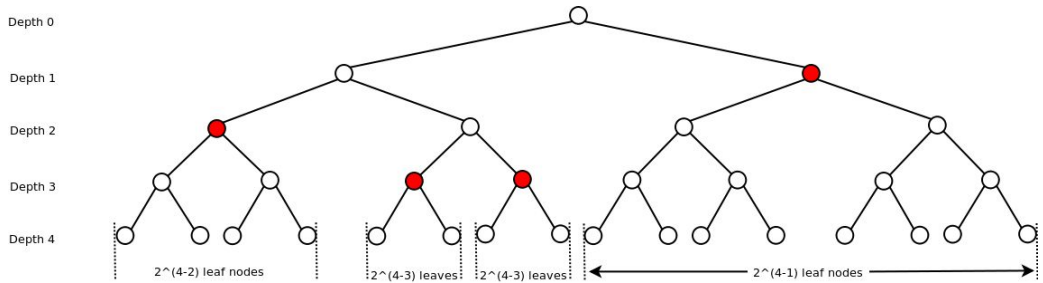
- Part II: For any set of lengths L, if $\sum_{l\in L} 2^{-l} \leq 1$

then there is a prefix code C of the same size such that l(wi) = li (i=1,..., |L|)

Kraft-McMillan Inequality Part I: Proof for Prefix Codes

- Part I: For any prefix code C, we have that: $\sum_i 2^{-l_i} \leq 1$ where li is the length of codeword i.
- Proof (sketch):

$$l_1 \leq l_2 \leq \cdots \leq l_n$$



Prefix Code → Tree where codewords are leaves → Expand into full tree of depth l_n

Codeword i "covers" $2^{l_n - l_i}$ leaves of the expanded tree.

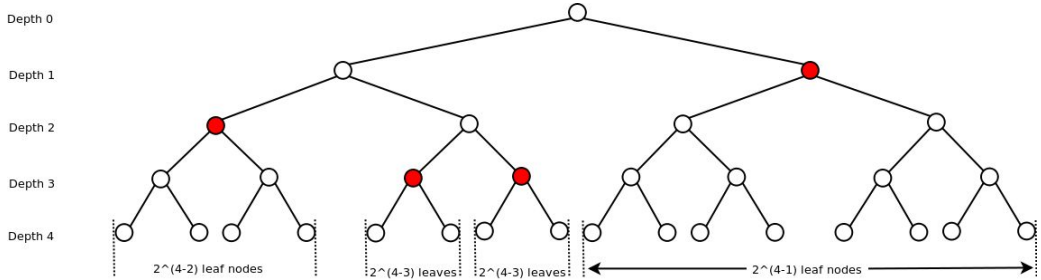
There is no overlap in leaves covered → $\sum_i 2^{l_n - l_i} \leq 2^{l_n} \rightarrow \sum_i 2^{-l_i} \leq 1$

Kraft-McMillan Inequality Part II: Proof

- Part II: For any set of lengths li, if $\sum_i 2^{-l_i} \leq 1$ then there is a prefix code C of the same size such that l(wi) = li (i=1,..., |L|)

- Proof (sketch):

$$l_1 \leq l_2 \leq \cdots \leq l_n$$



Consider full tree of depth l_n

For each i, pick any node of depth l_i still available, "covering" $2^{l_n - l_i}$ leaves of the expanded tree.

Is there enough space in the tree? Yes, if $\sum_i 2^{l_n - l_i} \leq 2^{l_n}$, which holds per: $\sum_i 2^{-l_i} \leq 1$

Consequence: Shannon Theorem 1948

- For any message distribution P(X) and associated uniquely decodable code C:

$$H(X) \leq l_a(C)$$

- Proof:

$$\begin{aligned}
H(X) - l_a(C) &= \sum_i p(x_i) \log_2 \frac{1}{p(x_i)} \\
&= \sum_i p(x_i) \left(\log_2 \frac{1}{p(x_i)} - l_i \right) \\
&= \sum_i p(x_i) \left(\log_2 \frac{1}{p(x_i)} - \log_2 2^{l_i} \right) \\
&= \sum_i p(x_i) \log_2 \frac{2^{-l_i}}{p(x_i)} \\
&\leq \log_2 \left(\sum_i 2^{-l_i} \right) \\
&\leq 0
\end{aligned}$$

Converse: Optimal Prefix Code is within 1 of H(X)

- Can we have a prefix code with code lengths $l_i = \lceil \log_2 \frac{1}{p(x_i)} \rceil$?

$$\begin{aligned}\sum_i 2^{-l_i} &= \sum_i 2^{-\lceil \log_2 \frac{1}{p(x_i)} \rceil} \\ &\leq \sum_i 2^{-\log_2 \frac{1}{p(x_i)}} \\ &= \sum_i p(x_i) \\ &= 1\end{aligned}$$

→ yes! Per Kraft-McMillan there is a prefix code with these lengths

- Now, what's the expected code length for this code?

$$\begin{aligned}l_a &= \sum_i p(x_i) l_i = \sum_i p(x_i) \lceil \log \frac{1}{p(x_i)} \rceil \\ &\leq \sum_i p(x_i)(1 + \log \frac{1}{p(x_i)}) \\ &= 1 + \sum_i p(x_i) \log \frac{1}{p(x_i)} \\ &= 1 + H(X)\end{aligned}$$

Huffman Codes are optimal prefix codes

Proof (sketch): by induction on number of symbols n

- Call the two lowest probability symbols x and y
- Optimal prefix codes have two leaves in every lowest level branch

  (otherwise, could do one less split)
- W.l.o.g. can have x and y have the same parent
- $\longrightarrow$ every optimal prefix tree has x and y together at lowest level with same parent
- No matter the tree structure, the additional cost of having x and y (rather than simply a parent symbol z) = $p(x) + p(y)$
- The n symbol Huffman code tree adds this minimal extra cost to the optimal n-1 symbol Huffman code tree (which is optimal by induction)

Quick Recap

Entropy = expected encoding length when encoding each symbol i with $\log_2 \frac{1}{p(x_i)}$ bits

$$H(X) = \sum_i p(x_i) \log_2 \frac{1}{p(x_i)}$$

Theorem [Shannon 1948] If data source P(X) is an order 0 Markov model, then any compression scheme must use >= H(X) bits per symbol on average

Theorem [Huffman 1952] If data source P(X) is an order 0 Markov model, then the Huffman code uses <= H(X) + 1 bits per symbol on average

Entropy of the English Language

Q. How much information is in each character of the English language?

Q. How can we measure it?

model = English text

A. [Shannon's 1951 experiment]

- Asked subjects to predict next character given previous text.
- The number of guesses required for right answer:

# of guesses	1	2	3	4	5	≥ 6
Fraction	0.79	0.08	0.03	0.02	0.02	0.05

- Shannon's estimate: about 1 bit per char [0.6 - 1.3].

Compression less than 1 bit/char for English ? If not, keep trying!

Outline

- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
  - Fixed length
  - Variable length
  - Huffman
- Theoretical limits
  - Shannon
  - Kraft-McMillan
  - Huffman codes
- ***Coding Considerations***
  - ***What happens if model \hat{p} only approximation of p?***
  - ***Higher order models***
  - ***+1***
- Arithmetic coding
- Lempel-Ziv

What if we use $\hat{p}$ to approximate p ?

Expected code length when using $\hat{p}$ to construct code:

$$
\begin{aligned}
l_a &= \sum_i p_i \log \frac{1}{\hat{p}_i} \\
&= \sum_i p_i \log \frac{1}{\hat{p}_i} - \sum_i p_i \log \frac{1}{p_i} + \sum_i p_i \log \frac{1}{p_i} \\
&= \sum_i p_i \log \frac{p_i}{\hat{p}_i} + \sum_i p_i \log \frac{1}{p_i} \\
&= KL(p \| \hat{p}) + H(p)
\end{aligned}
$$

→ **pay price = KL(.||.)**

Recall: KL(. || .) >=0

Proof: $KL(p\|q) = \sum_i p_i \log \frac{p_i}{q_i} = \sum_i p_i - \log \frac{q_i}{p_i} \geq \log \left(\sum_i p_i \frac{q_i}{p_i} \right) = \log 1 = 0$

What if P(X) has high entropy?

High entropy would mean long code length...

Decrease the entropy by considering P(X|C):

$$H(X|C) = \sum_c p(c) \sum_x p(x|c) \log_2 \frac{1}{p(x|c)}$$

Conditional entropy is smaller:

$$H(X|C) \leq H(X) = \sum_c p(c) \sum_x p(x|c) \log_2 \frac{1}{p(x)}$$

Note: Autoregressive models do this!

+1 ?

Theorem [Huffman 1952] If data source P(X) is an order 0 Markov model, then the Huffman code uses <= H(X) + 1 bits per symbol on average

Q: Sounds great, but is it great? Or might it still be pretty bad?

A: if we have good predictive models which make H(X) very low, then this can be very high overhead

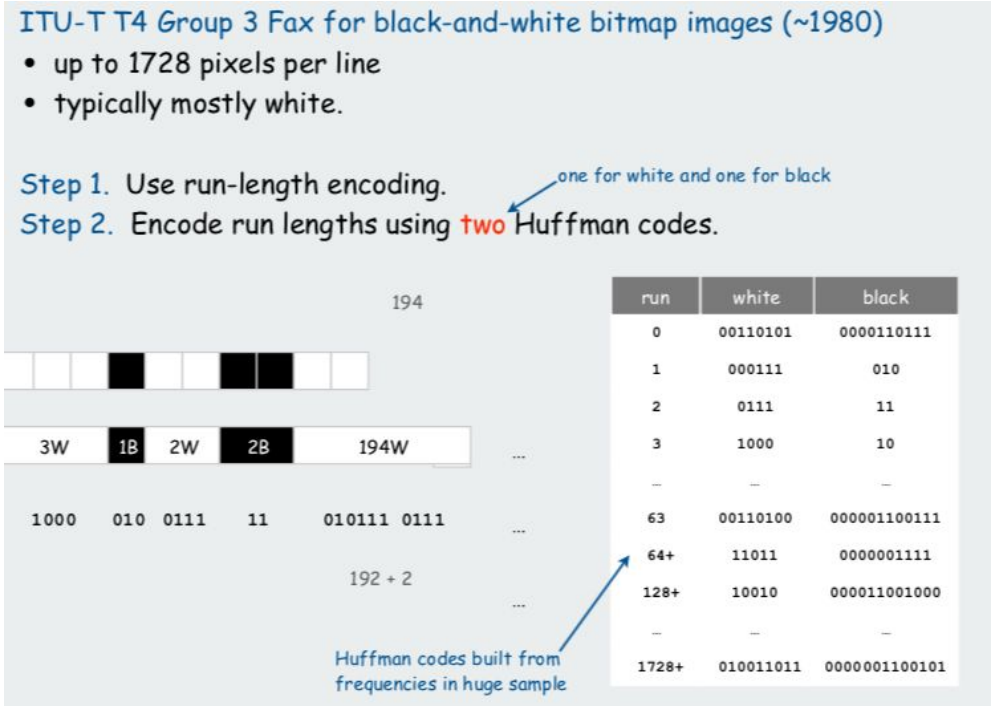
E.g.: P(X) = {0.9, 0.05. 0.05} → H(X) = 0.569

Best (per symbol) code: {0, 11, 10} → expected code length = 0.9 +.1+.1 = 1.1

→ almost double!

Run-length Coding

- Encode many symbols together, pay the +1 less frequently.
  E.g. Fax



Entropy of the English Language

	bits/char
bits $\lceil \log(96) \rceil$	7
entropy	4.5
Huffman Code (avg.)	4.7
Entropy (Groups of 8)	2.4
Asymptotically approaches:	1.3
Compress	3.7
Gzip	2.7
BOA	2.0

Table 1: Information Content of the English Language

[Calgary Corpus; Belloch 2013]

Outline

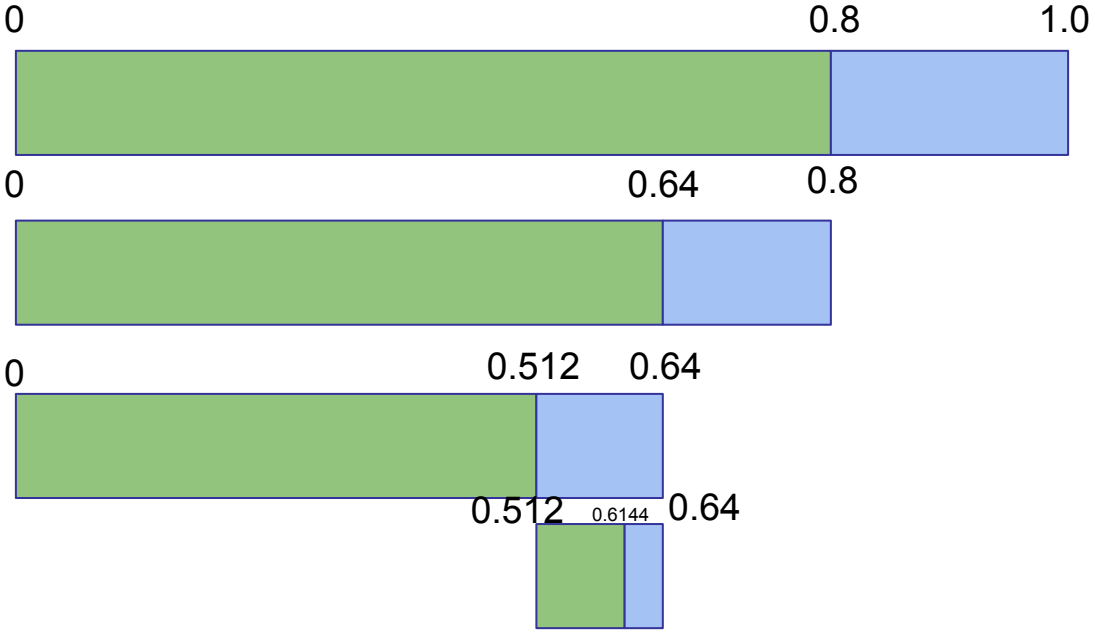
- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
  - Fixed length
  - Variable length
  - Huffman
- Theoretical limits
  - Shannon
  - Kraft-McMillan
  - Huffman codes
- Coding Considerations
  - What happens if model \hat{p} only approximation of p?
  - Higher order models
  - +1
- ***Arithmetic coding***
- Lempel-Ziv

Arithmetic Coding

- Key motivation: flexible system to encode multiple symbols in one go to avoid the potential “+1” overhead on every symbol
- Key idea: encode through indexing into distribution
- PS: very compatible with autoregressive models! :)

Simple Arithmetic Coding Example

P(a) = 0.8; P(b) = 0.2; let's encode: "aaba"



→ send interval [0.512, 0.6144)

How to encode an interval?

Naive attempt:

Represent each interval by selecting the number within the interval which has the fewest bits in binary fractional notation, and use that as the code.

E.g. if we had the intervals [0, .33), [.33, 67), and [.67, 1) we would represent these with .01(1/4), .1(1/2), and .11(3/4). It is not hard to show that for an interval of size s we need at most $-\lceil \log_2 s \rceil$ bits to represent such a number. The problem is that these codes are not a set of prefix codes.

Instead:

Have each binary number correspond to interval of all possible completions.

So for above example: .00 [0, .25), .100 [.5, .625), .11 [.75, 1)

For interval of size s can always find a codeword of length $-\lceil \log_2 s \rceil + 1$

$\rightarrow$ code length <= H(X) + 2 [+1 from rounding up, +1 from the +1] --- *+2 overhead only once for full sequence X*

Any remaining challenges?

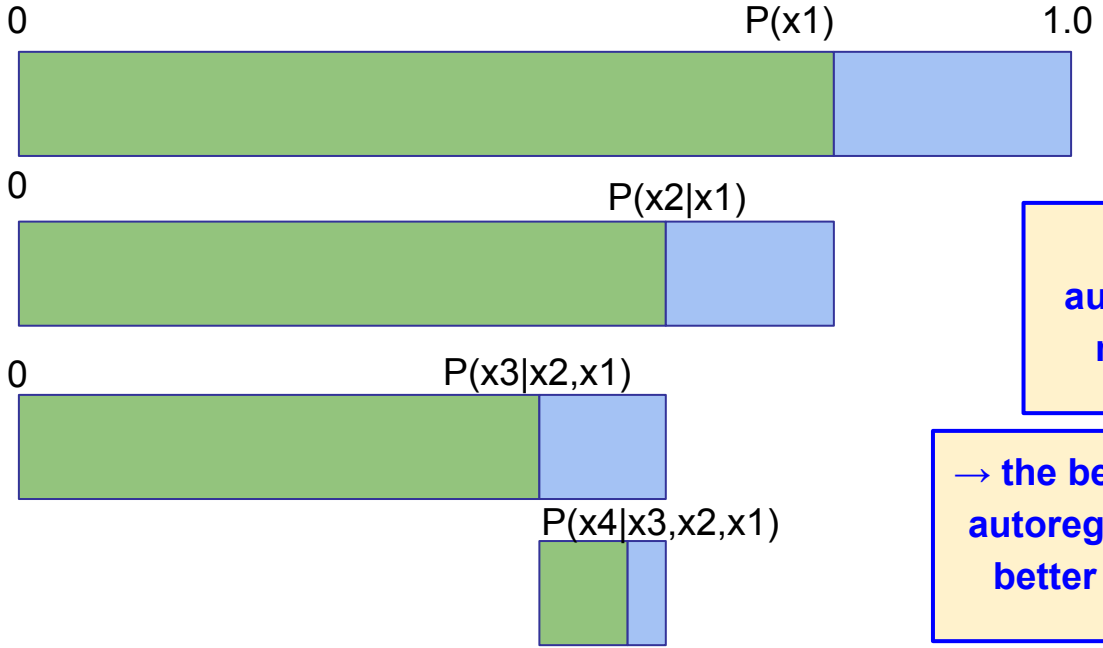
- Interval could keep straddling (e.g.) 0.5, in which case can't send the first bit till done
  - Solution: split message into blocks

- Assumes infinite precision
  - Solution: integer implementation (see Blelloch 2013)

Arithmetic Coding with Autoregressive Models

Recall: P(a) = 0.8; P(b) = 0.2; let's encode: "aaba"

BUT: no need for p(.) to be the same everywhere



→ use autoregressive model for p

→ the better log-prob of the autoregressive model, the better the compression

Outline

- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
  - Fixed length
  - Variable length
  - Huffman
- Theoretical limits
  - Shannon
  - Kraft-McMillan
  - Huffman codes
- Coding Considerations
  - What happens if model \hat{p} only approximation of p?
  - Higher order models
  - +1
- Arithmetic coding
- ***Lempel-Ziv***

Types of Statistical Methods

Static model. Same model for all texts.
- Fast.
- Not optimal: different texts have different statistical properties.
- Ex: ASCII, Morse code. Huffman code (generic p).

Dynamic model. Generate model based on text.
- Preliminary pass needed to generate model.
- Must transmit the model.
- Ex: Huffman code (p fitted to entire text).

Adaptive model. Progressively learn and update model as you read text.
- More accurate modeling produces better compression.
- Decoding must start from beginning.
- Ex: LZW. (~ “p” fitted to text seen so far)

General structure of LZ algorithms

- Given a position in a file, look through the preceding part of the file to find the longest match to the string starting at the current position, and output some code that refers to that match.

- Variations:
  - How far back to look (sliding window)
  - When to add strings to the dictionary

LZ77

(figure source: Belloch 2013; see Blelloch 2013 for other LZ(W) variants)

Step	Input String															Output Code
1	[a]	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(0, 0, a)
2	**a**	[a]	c	a	a	c	a	b	c	a	b	a	a	a	c	(1, 1, c)
3	**a**	**a**	**c**	[a]	a	c	a	b	c	a	b	a	a	a	c	(3, 4, b)
4	a	a	**c**	**a**	**a**	**c**	**a**	**b**	[c]	a	b	a	a	a	c	(3, 3, a)
5	a	a	c	a	a	c	**a**	**b**	**c**	**a**	**b**	**a**	[a]	a	c	(1, 2, c)

Figure 12: An example of LZ77 with a dictionary of size 6 and a lookahead buffer of size 4. The cursor position is boxed, the dictionary is bold faced, and the lookahed buffer is underlined. The last step does not find the longer match (10,3,1) since it is outside of the window.

Compression Challenge

https://www.compression.cc/challenge/