CS294-158 Deep Unsupervised Learning

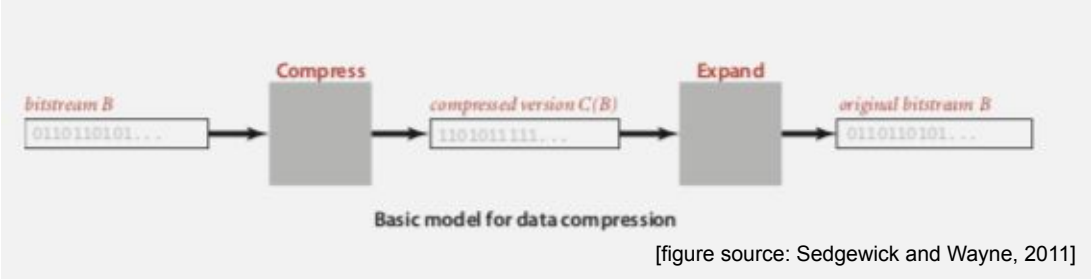
Lecture 10 Compression

Pieter Abbeel, Xi (Peter) Chen, Jonathan Ho, Aravind Srinivas, Alex Li, Wilson Yan

UC Berkeley

Compression: What and Why?

- What?
  - Reduce number of bits for encoding a message (e.g. image, speech, mus



[figure source: Sedgewick and Wayne, 2011]

- Why?
  - Save time/bandwidth/space when transmitting/storing
  - AI: ability to compress data reflects understanding of the data

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

http://prize.hutter1.net/

500'000€ Prize for Compressing Human Knowledge

(widely known as the Hutter Prize)

Compress the 1GB file `enwik9` to less than the current record of about 116MB

News: *The H-Prize went BIG! Why?*
10×*Size*, 10×*Purse*, 10×*Time*, 10×*RAM*, 10×*HDD*.

Who will get here first?

Being able to compress well is closely related to intelligence as explained below. While intelligence is a slippery concept, file sizes are hard numbers. Wikipedia is an extensive snapshot of Human Knowledge. If you can compress the first 1GB of Wikipedia better than your predecessors, your (de)compressor likely has to be smart(er). The intention of this prize is to encourage development of intelligent compressors/programs as a path to AGI.

Interview with Lex Fridman (26.Feb'20) (Video, Audio, Tweet)

The Task

Losslessly compress the 1GB file `enwik9` to less than 116MB. More precisely:

- Create a Linux or Windows compressor `comp.exe` of size *S1* that compresses `enwik9` to `archive.exe` of size *S2* such that $S:=S1+S2 < L := 116'673'681$ = previous record.
- If run, `archive.exe` produces (without input from other sources) a 10^9 byte file that is identical to `enwik9`.
- If we can verify your claim, you are eligible for a prize of 500'000€×(1-*S*/*L*). Minimum claim is 5'000€ (1% improvement).
- Restrictions: Must run in ≤100 hours using a single CPU core and <10GB RAM and <100GB HDD on our test machine.

Remark: You can download the zipped version `enwik9.zip` of `enwik9` here (≈300MB). Please find more details including constraints and relaxations at http://prize.hutter1.net/hrules.htm.

http://www.compression.cc/challenge/

CLIC

WORKSHOP CHALLENGE CALL FOR PAPERS ABOUT PREVIOUS CHALLENGES

CHALLENGE

Motivation

The domain of image compression has traditionally used approaches discussed in forums such as ICASSP, ICIP and other very specialized venues like PCS, DCC, and ITU/MPEG expert groups. This workshop and challenge was the first computer-vision event to explicitly focus on these fields. Many techniques discussed at computer-vision meetings have relevance for lossy compression. For example, super-resolution and artifact removal can be viewed as special cases of the lossy compression problem where the encoder is fixed and only the decoder is trained. But also inpainting, colorization, optical flow, generative adversarial networks and other probabilistic models have been used as part of lossy compression pipelines. Lossy compression is therefore a potential topic that can benefit a lot from a large portion of the CVPR community.

Recent advances in machine learning have led to an increased interest in applying neural networks to the problem of compression. At CVPR 2017, for example, one of the oral presentations discussed compression using recurrent convolutional networks. In recent CVPRs, multiple lossy and lossless compression works were presented. In order to foster more growth in this area, this workshop not only encourages more development but also establishes baselines, educates, and proposes a common benchmark and protocol for evaluation. This is crucial, because without a benchmark, a common way to compare methods, it will be very difficult to measure progress.

We propose hosting a lossy image and video compression challenge which specifically targets methods which have been traditionally overlooked, with a focus on neural networks (but also welcomes traditional approaches). Such methods typically consist of an encoder subsystem, taking images/videos and producing representations which are more easily compressed than pixel representations (e.g., it could be a stack of convolutions, producing an integer feature map), which is then followed by an arithmetic coder. The arithmetic coder uses a probabilistic model of integer codes in order to generate a compressed bit stream. The compressed bit stream makes up the file to be stored or transmitted. In order to decompress this bit stream, two additional steps are needed: first, an arithmetic decoder, which has a shared probability model with the encoder. This reconstructs (losslessly) the integers produced by the encoder. The last step consists of another decoder producing a reconstruction of the original images/videos.

In the computer vision community, many authors will be familiar with a multitude of configurations which can act as either the encoder or the decoder, but probably few are familiar with the implementation of an arithmetic coder/decoder. As part of our previous challenge, we therefore released a reference arithmetic coder/decoder in order to allow researchers to focus on the parts of the system for which they are experts. For the 2nd edition, we released a drop-in range coder/decoder in Tensorflow.

While having a compression algorithm is an interesting feat by itself, it does not mean much unless the results it produces compare well against other similar algorithms and established baselines on realistic benchmarks. In order to ensure realism, we have collected a set of images which represent a much more realistic view of the types of images which are widely available (unlike the well established benchmarks which rely on the images from the Kodak PhotoCD, having a resolution of 768x512, or Tecnick, which has images of around 1.44 megapixels). We will also provide the performance results from current state-of-the-art compression systems as baselines, like WebP and BPG. For the P-frame track, we will use an existing dataset which we will provide preprocessed to allow for easier training. Additionally, we will use an existing video dataset for evaluation and will not be creating new video content for this challenge.

Prizes

TBA

Compression: Why in this course?

- For a substantial part of this course we studied generative models

- Turns out that compression utilizes generative models, and the better the model, the better the compression can be

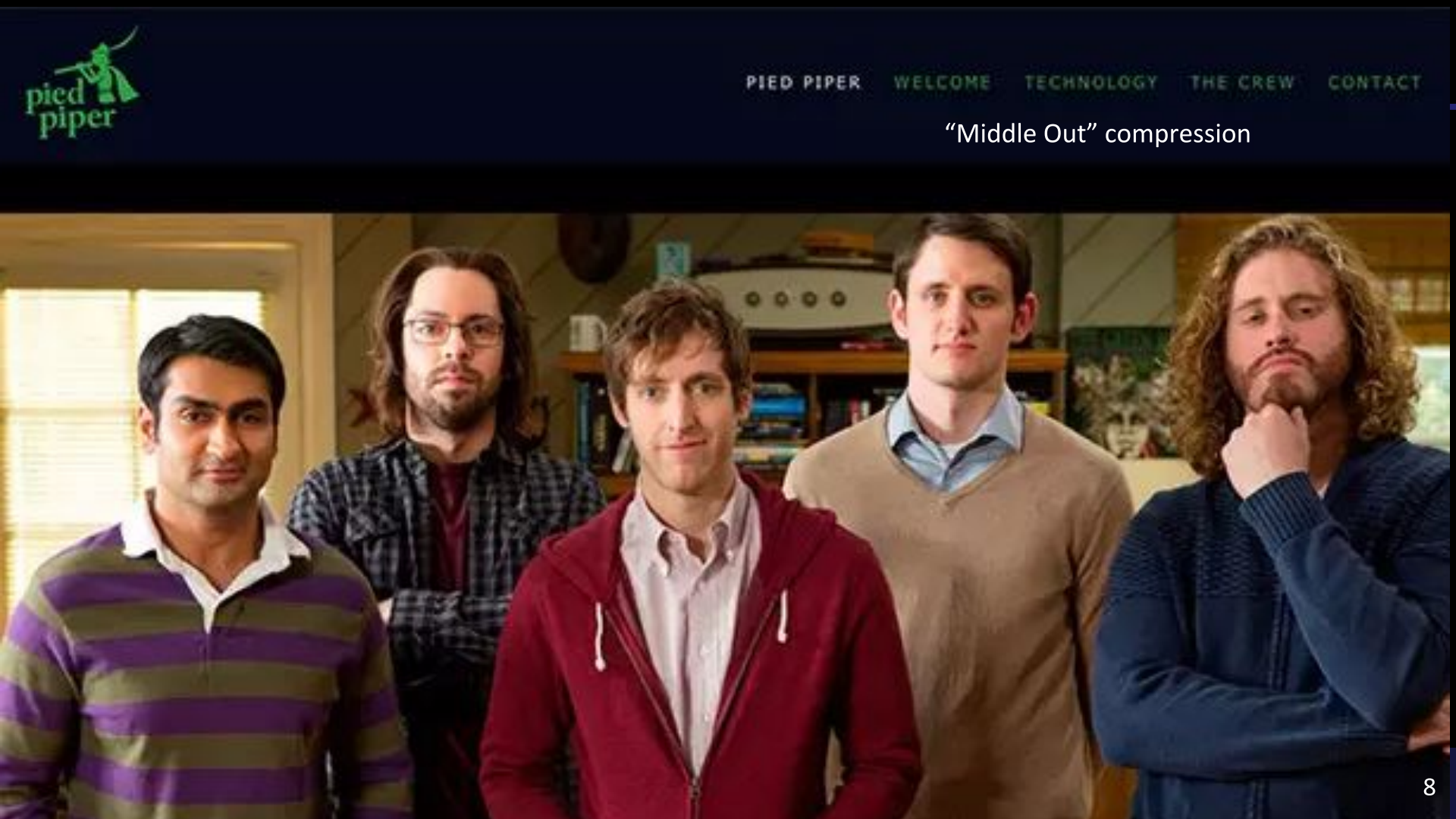
Reading Material

Core reading:

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

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



"Middle Out" compression

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.

Why compression possible?

Example statistical pattern in text:

"... 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."

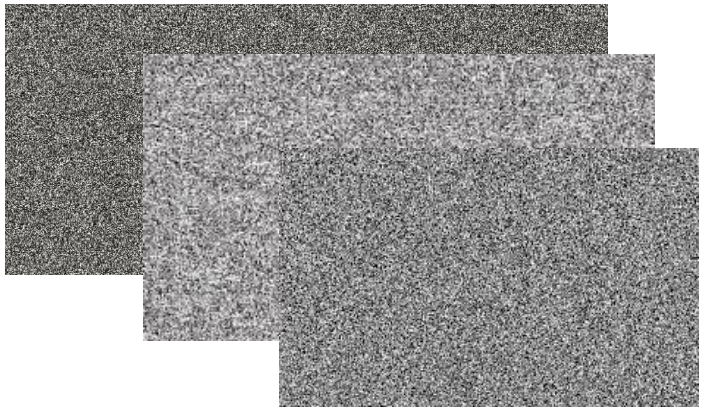
--- Graham Rawlinson, PhD Thesis 1976, Nottingham U.

Why compression possible?

Dataset 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
- Coding Considerations
- Autoregressive Models and Arithmetic coding
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- Flow Models
- Lempel-Ziv

Naive Coding: Fixed Length (***no*** compression!)

E.g. ASCII: 7 bits per character

USASCII code chart

Bits b_4	b_3	b_2	b_1	Row ↓ / Column → 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

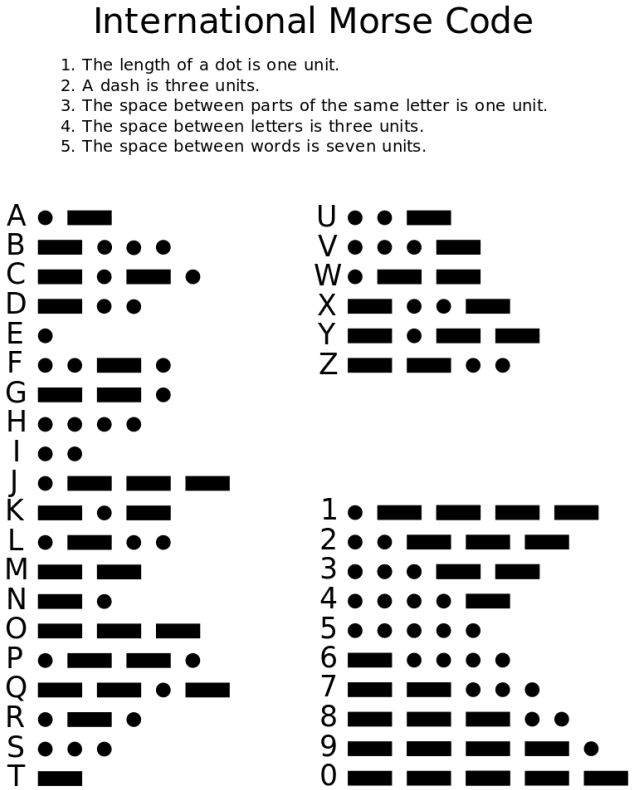
Q: How do we avoid ambiguity?

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

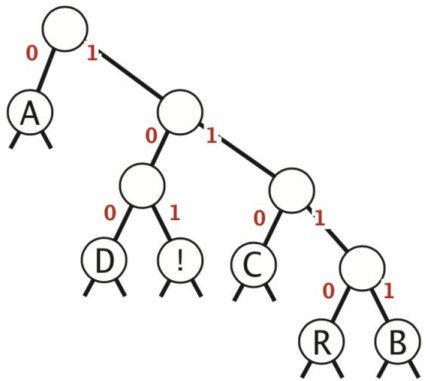
Symbols in leaves

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

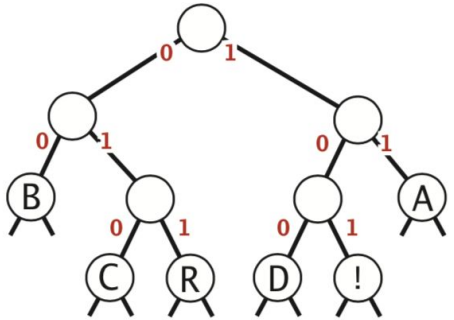
0111111100110010001111111100101 ← 30 bits

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:

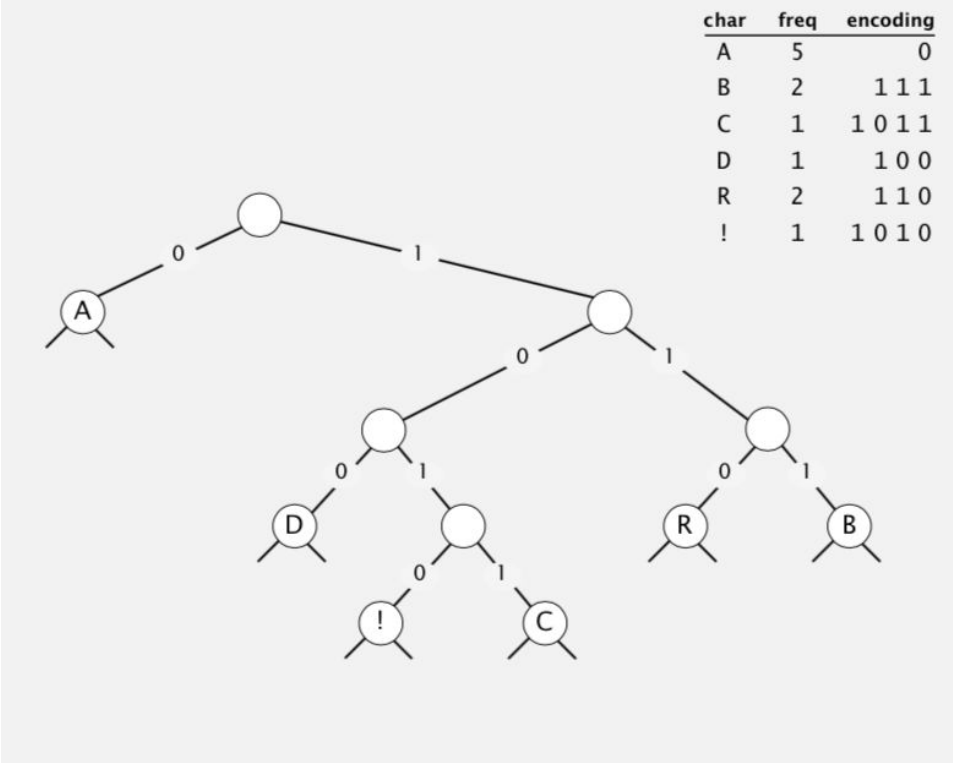
- Consider probability p[i] of each symbol i in the input
- Start with one node corresponding to each symbol i (with weight p[i])
- Repeat until single trie formed:
  - Select two tries with min probabilities p[k] and p[l]
  - Merge into single trie with probability p[k]+p[l]

[Why optimal? See later in lecture]

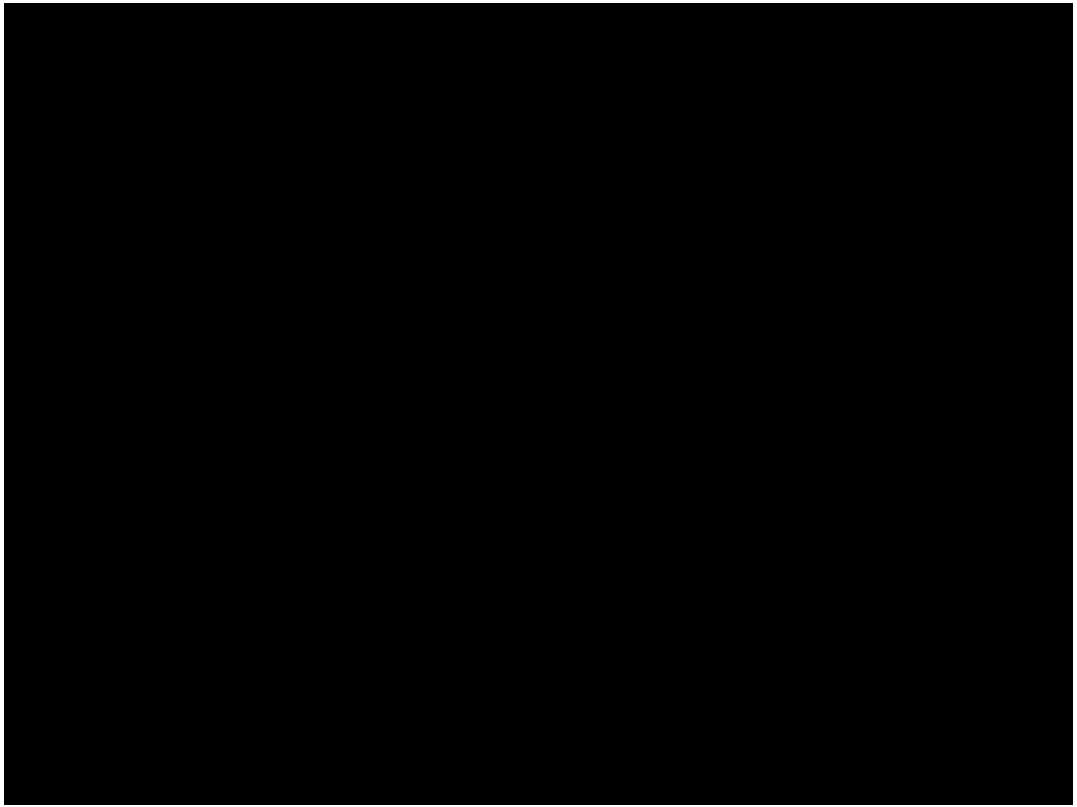
Huffman Codes -- Example

A	0.20
B	0.10
C	0.05
D	0.21
E	0.36
F	0.08

Huffman Codes -- Example



Huffman Codes: Demo



[video source: Sedgewick and Wayne, 2011]

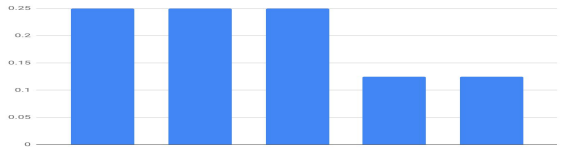
Huffman Code -- Applications

Outline

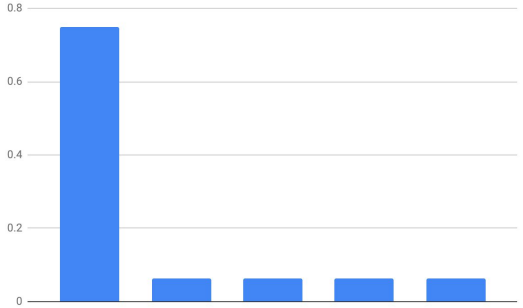
- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
- ***Theoretical limits***
  - ***Shannon***
  - ***Kraft-McMillan***
  - ***Huffman codes***
- Coding Considerations
- Autoregressive Models and Arithmetic coding
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- Flow Models
- Lempel-Ziv

Shannon: Entropy = measure of information

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



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

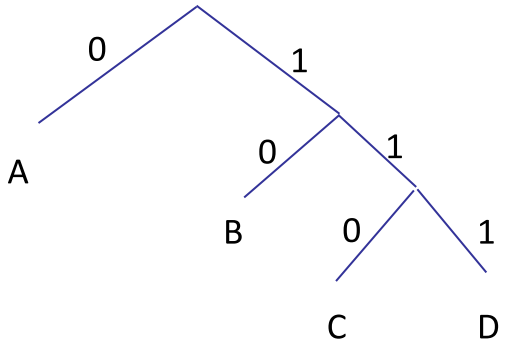


$$\begin{aligned} p(s) &= \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\} \\ H &= 0.75 \times \log_2(\frac{4}{3}) + 4 \times 0.0625 \times \log_2 16 \\ &= 0.3 + 1 \\ &= 1.3 \end{aligned}$$

Entropy coding

- Set code length $l_i = \lceil \log_2 \frac{1}{p(x_i)} \rceil$

- Example: (A, ½), (B, ¼), (C, ⅛), (D, ⅛)



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.
I.e.: lengths l(w) have to be large enough to be uniquely decodable

- 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|)

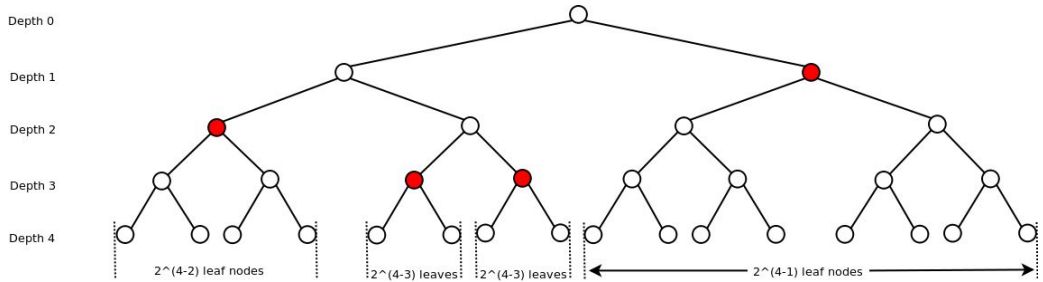
→ we never need to resort to a non-prefix code

Kraft-McMillan Inequality Part I: Proof (for special case of 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.

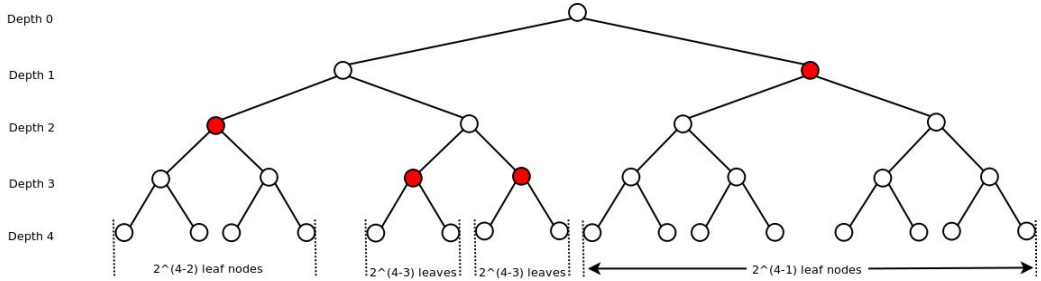
There is no overlap in leaves covered → $\sum_i 2^{l_n - l_i} \leq 2^{l_n} \rightarrow \sum_i 2^{-l_i} \leq 1$

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:

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

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

Outline

- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
- Theoretical limits
- ***Coding Considerations***
  - ***What happens if model \hat{p} only approximation of p?***
  - ***Higher order models***
  - ***+1***
- Autoregressive Models and Arithmetic coding
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- Flow Models
- 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(.||.)
→ log likelihood objective for learning $\hat{p}$ is aligned with compression

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

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

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] 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

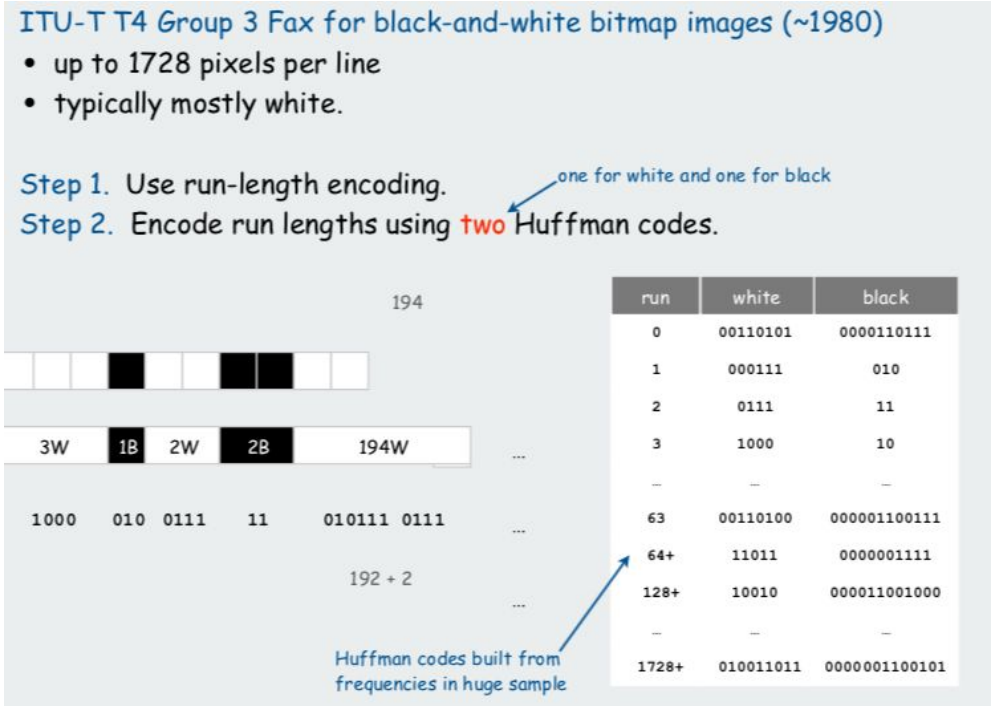
E.g.: P(X) = {0.9, 0.05. 0.05} → H(X) = 0.569

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

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!

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
- Theoretical limits
- Coding Considerations
- ***Autoregressive Models and Arithmetic coding***
  - ***Arithmetic Coding***
  - ***Arithmetic Coding with AR Models***
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- Flow Models
- Lempel-Ziv

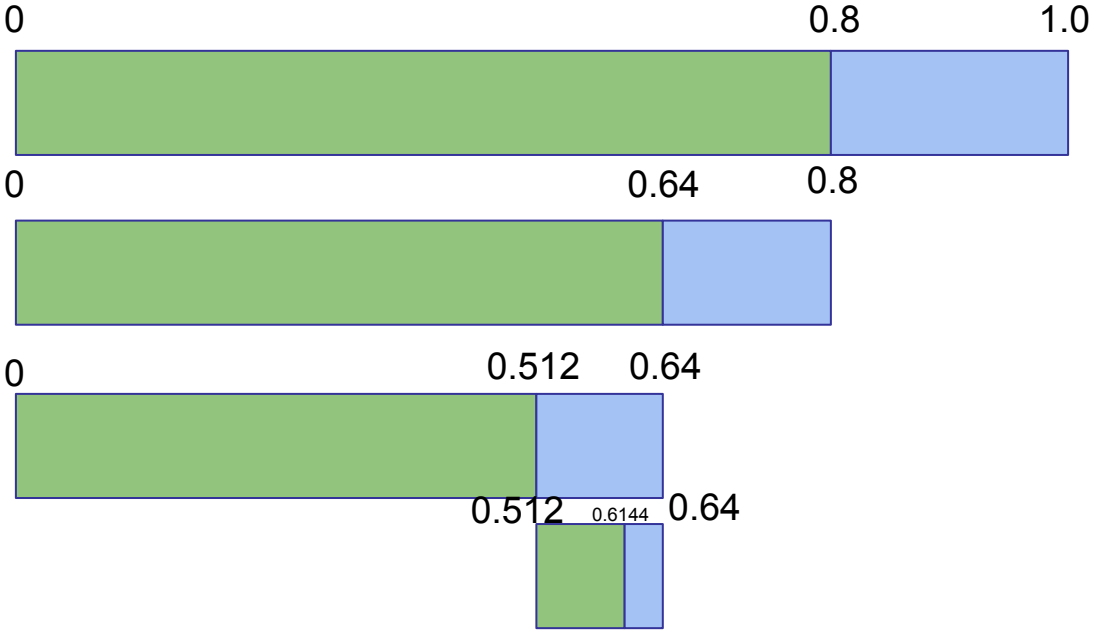
Arithmetic Coding

- Key motivation: flexible system to group multiple symbols to avoid the potential "+1" overhead on every symbol
- How many symbols? Which symbols to group?
- Key idea: encode through indexing into distribution
  - No need to decide how many symbols or which symbols
  - Works streaming over the data
- PS: very compatible with autoregressive models! :)

[Rissanen, Jorma J. *"Generalized Kraft inequality and arithmetic coding."* IBM Journal of research and development 20.3 (1976): 198-203.]

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 ⌈- log2 s⌉ 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 ⌈- log2 s⌉ + 1

→ code length <= H(X) + 2 [+1 from rounding up, +1 from the +1]

Note: *+2 overhead only once for full sequence X*

Any remaining challenges?

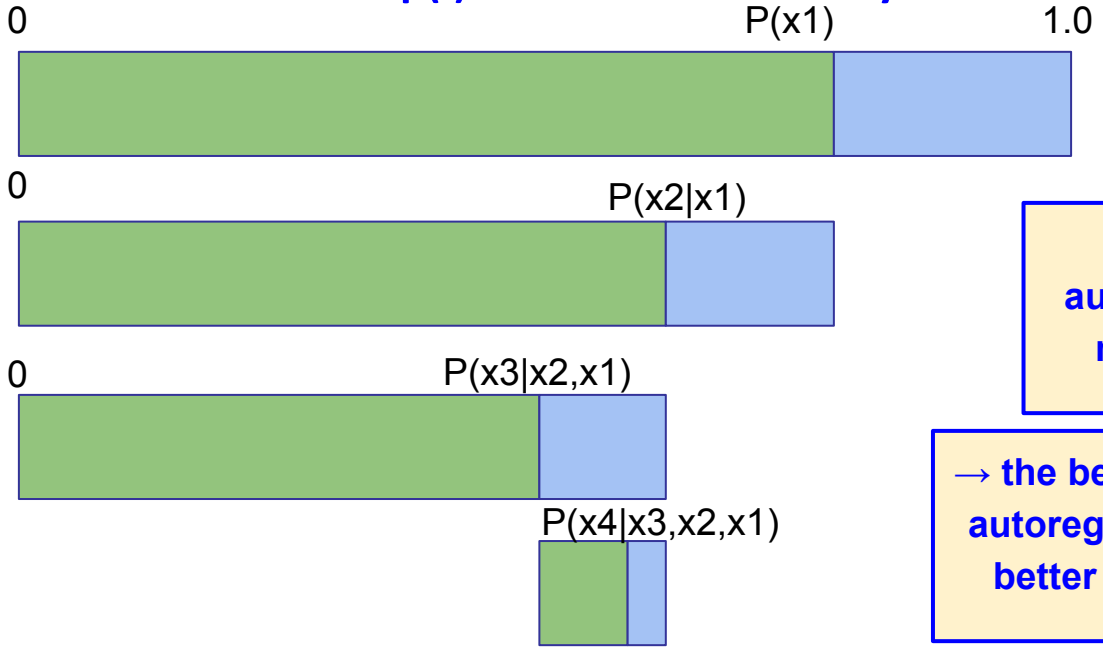
- Interval could keep straddling (e.g.) 0.5, in which case can't send the first bit till done
  - Solution: split message into blocks

- 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
- Theoretical limits
- Coding Considerations
- Autoregressive Models and Arithmetic coding
- ***VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)***
  - ***Continuous x***
  - ***Mixture models***
  - ***Bits-back coding***
  - ***BB-ANS***
  - ***Bit-Swap***
  - ***ANS***
- Flow Models
- Lempel-Ziv

References for this lecture

- Brendan J. Frey and Geoffrey E. Hinton, *"Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model,"* The Computer Journal 1997

- James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,"* ICLR 2019

- Friso Kingma, Pieter Abbeel, Jonathan Ho, *"Bit-Swap: Practical Bits Back Coding with Hierarchical Latent Variables,"* ICML 2019

- Jarek Duda, *"Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding,"* 2007

- GE Hinton, D Van Camp, *"Keeping Neural Networks Simple by Minimizing the Description Length of the Weights"*, 1993

Refresher on Coding

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

Arithmetic Coding (AC): Pays only +2 for entire batch of symbols.

Key Assumptions

Previous slide ***assumes*** we have a model p(x) for which we can:

- Huffman: tractably enumerate all x
- AC: tractably assign intervals to each x

→ Generically: small number of possible values x can take on

When violated?

- When x continuous --- but easily "fixable" (see next slide)
- When x high-D --- challenge we need to address!
  - Key observation:
    - some high-D distributions still allow for convenient coding
    - leverage that to efficiently code mixture models of easy high-D distributions
      key win: mixture models much more expressive then individual components!

Continuous domain for X

A real number x has infinite information, so we can't expect to send x exactly

→ assume discretization with some precision t

→ can discretize in x domain (or, alternatively, in the cumulative distribution domain)

E.g. X ~ N(0,1):

Differential Entropy <> Entropy of discretized variable

$$
\begin{aligned}
H(x_{\text{disc,t}}) &= -\sum_i tp(x_i)\log(tp(x_i)) \\
&= -\sum_i tp(x_i)\log p(x_i) - \sum_i tp(x_i)\log t \\
&\approx -\int_x p(x)\log p(x)dx - \log t \int_x p(x)dx \\
&= h(x) - \log t
\end{aligned}
$$

-> matches intuition: making t smaller results in more buckets hence higher entropy

→ can use Huffman coding and AC again

Some high-dimensional P(x) allow for easy coding

X is high dimensional, but P(X) sometimes still easy to encode

- E.g.
  - X ~ N(0,I)
    - can treat as n independent random variables (n = dimensionality of x)
    - each independent variable can be coded efficiently (per previous slide)
  - X autoregressive
    - (see previous lecture)

Mixture Models

Mixture models allow to represent wider range of distributions than their components

E.g. mixture of Gaussians = much more complex distribution than a single Gaussian

Key Question

If P(X) is a mixture model of easily encodable distributions, can we efficiently encode P(X)?

Reminders:

* We'll make our illustrations in 1-D to get the point across in a way that's easy to draw on slides, but keep in mind we are covering a method that generalizes to high-D.

* We won't allow ourselves to rely on domain of X being small, in which case the solution is trivial by simply calculating the marginal for each value X can take on, summed over all mixture components and then using Huffman or AC.

Alright, now let's see if we can handle these harder cases!

Mixture Model Running Example

$$P(x) = \sum_i p(i)p(x|i)$$

$p(x|i)$ assumed easy to code for

→ will cost: $\log \frac{1}{p(x|i)}$

Scheme 1: “Max-Mode” Coding

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

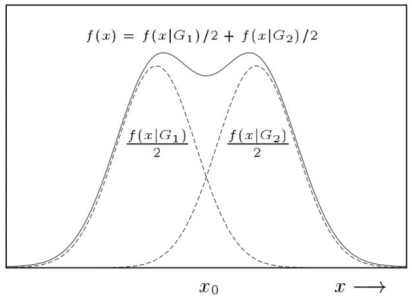
- Find i that maximizes p(i | x) (x is fixed, so also max p(i,x))
- Send i --- cost: log 1/p(i)
- Send x --- cost: log 1/p(x|i)

Expected code length:

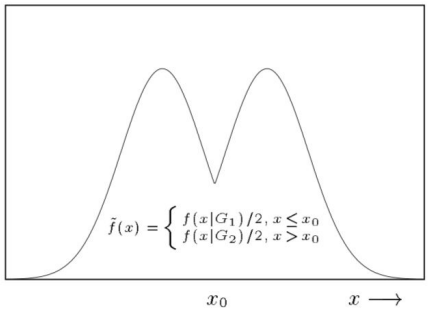
$$l_a(C) = \sum_x p(x) \min_i \log \frac{1}{p(i,x)}$$
$$= \sum_{j,x} p(j,x) \min_i \log \frac{1}{p(i,x)}$$

Optimal? NO --- because effectively using different distribution Q(x) $\rightarrow$ cost: H(X) + KL(P||Q)

P:



Q:



Scheme 2: “Posterior-Sampling” Coding

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

- Sample i from p(i|x)
- Send i --- cost: log 1/p(i) --- why p(i), why not p(i|x)? b/c recipient doesn’t have x
- Send x --- cost: log 1/p(x|i) --- this is probably efficient, but not as efficient as using best i...

Expected code length: $$l_a(C) = \sum_{i,x} p(i,x) \log \frac{1}{p(i,x)}$$

Optimal? Yes and No. Yes if we like to send (i, x), b/c we use log 1/p(x,i)

BUT: we are looking to send just x, so overhead of log 1/p(i|x)

This is actually more expensive than the max-mode scheme from the previous slide

Scheme 3: Bits-Back Coding

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

- Sample i from p(i|x)
- Send i --- cost: log 1/p(i) --- why p(i)? b/c recipient doesn't have x
- Send x --- cost: log 1/p(x|i) --- this is probably efficient, but not as efficient as using best i...

BITS-BACK IDEA (EXACT INFERENCE):

recipient decodes i, x + knows p(i|x)

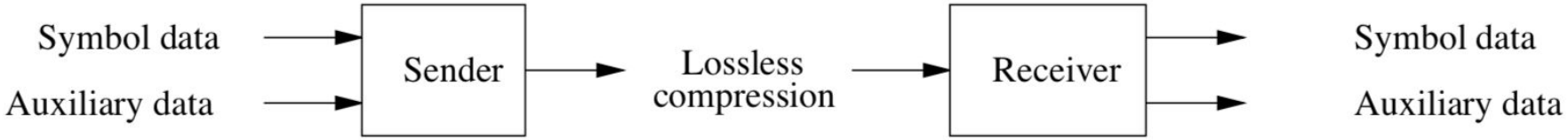
→ can reconstruct the random bits used to sample p(i|x)

→ those random bits were also sent → these are log 1/p(i|x) random bits, which we now don't have to count

→ bits-back coding cost: log 1/p(i) + log 1/p(x|i) - log 1/p(i|x) = log p(i|x) / (p(i) * p(x|i)) = log 1/p(x)

→ Optimal!!!

Bits-Back Coding in a Picture



Assumptions:
(1) We can compute $p(i|x)$
(2) We have auxiliary random data we like to transmit

Bits-Back Coding with Approximate Inference

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

- Sample i from q(i|x)
- Send i --- cost: log 1/p(i)
- Send x --- cost: log 1/p(x|i)

BITS-BACK IDEA (APPROXIMATE INFERENCE):

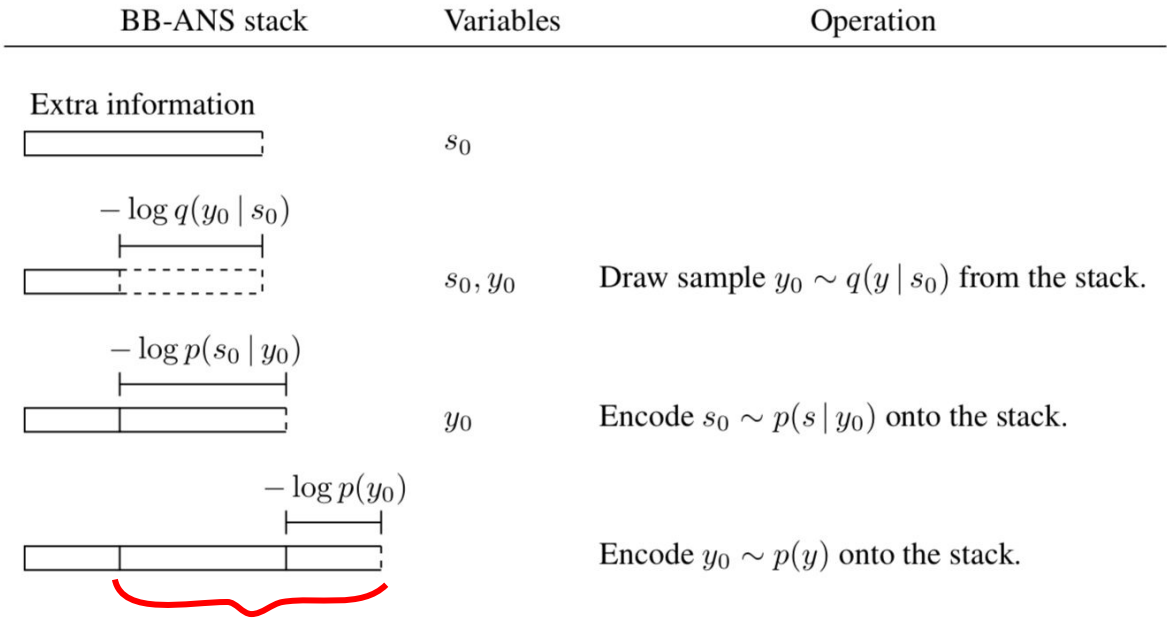
recipient decodes i, x + knows q(i|x)

→ can reconstruct the random bits used to sample q(i|x)

→ those random bits were also sent → these are log 1/q(i|x) random bits, which we now don't have to count

→ bits-back coding cost: log 1/p(i) + log 1/p(x|i) - log 1/q(i|x) = Evidence Lower Bound (ELBO) from VAE

How about that source of random bits we'd like to also send?



this encoding can serve as random bits for the next one

How do we actually get those bits back?

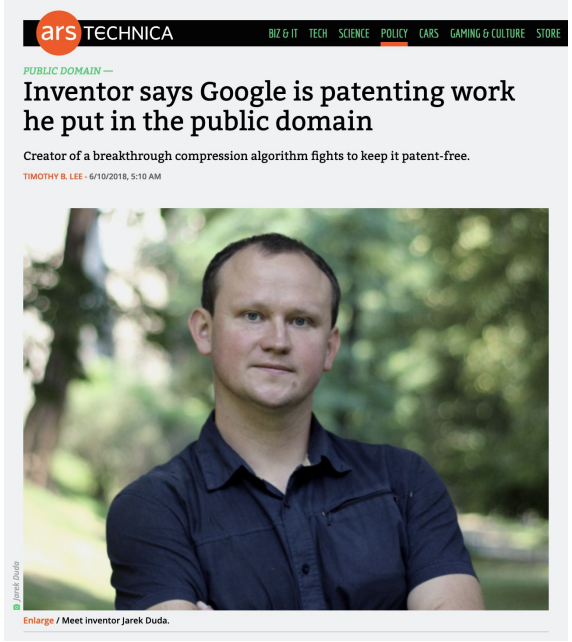
Recall we have: $i \sim q(i|x)$, and use random bits to generate i

How exactly can this work mechanically, and mechanics of getting them back?

A quick history of bits-back coding

- Wallace, 1990; Hinton & van Camp, 1993
  - Bits-back idea
- Frey & Hinton, 1996
  - Bits-back implementation with arithmetic coding; but not very natural fit
- Duda, 2009
  - Asymmetric Numeral Systems (ANS) coding
- Townsend, Bird & Barber, ICLR 2019
  - Bits Back with ANS (BB-ANS)
- F. Kingma, Abbeel & Ho, ICML 2019; Ho, Lohn, Abbeel, NeurIPS 2019
  - Bit Swap
  - Local Bits Back Coding

Asymmetric Numeral Systems (ANS)



Jarek Duda, *"Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding,"* 2007

Asymmetric Numeral Systems

- High level idea: use natural numbers as encodings, the longer the symbol sequence, the higher the natural number becomes

- Example: p(a) = 1/4 ; p(b) = 3/4
  - 1/4 of natural numbers should correspond to a symbol sequence ending in a
  - 3/4 _____ b

$$\mathbb{N} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \cdots\}$$

  - Encoding process: (encoding so far, new symbol) -> new encoding
    - (0,a) -> 0
    - (1,a) -> 4
    - (2,a) -> 8
    - (0,b) -> 1
    - (1,b) -> 2
    - (2,b) -> 3
    - (3,b) -> 5
    - (4,b) -> 6
  - General scheme:
    - (x, s) -> x / p(s) [approximately]

Asymmetric Numeral Systems

- High level idea: use natural numbers as encodings, the longer the symbol sequence, the higher the natural number becomes
- General scheme:
  - (x, s) -> x / p(s) [but made into an integer]
- Hence we end up with $$x_T \approx \frac{x_0}{p(s_1)p(s_2)p(s_3)\cdots p(s_T)}$$
- And code-length: $$\log x_T \approx C + \sum_t \log \frac{1}{p(s_t)}$$

Bits-back with ANS (BB-ANS)

Published as a conference paper at ICLR 2019

PRACTICAL LOSSLESS COMPRESSION WITH LATENT VARIABLES USING BITS BACK CODING

James Townsend, Thomas Bird & David Barber
Department of Computer Science
University College London
{james.townsend,thomas.bird,david.barber}@cs.ucl.ac.uk

ABSTRACT

Deep latent variable models have seen recent success in many data domains. Lossless compression is an application of these models which, despite having the potential to be highly useful, has yet to be implemented in a practical manner. We present 'Bits Back with ANS' (BB-ANS), a scheme to perform lossless compression with latent variable models at a near optimal rate. We demonstrate this scheme by using it to compress the MNIST dataset with a variational auto-encoder model (VAE), achieving compression rates superior to standard methods with only a simple VAE. Given that the scheme is highly amenable to parallelization, we conclude that with a sufficiently high quality generative model this scheme could be used to achieve substantial improvements in compression rate with acceptable running time. We make our implementation available open source at https://github.com/bits-back/bits-back.

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

BB-ANS

- Latent variable model with approximate posterior q

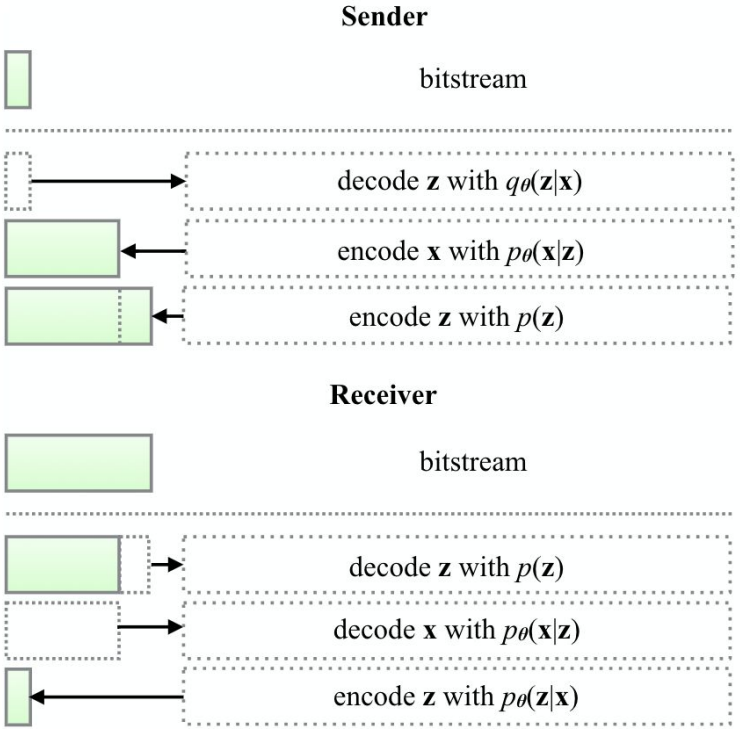
$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$$

- Encoding phase:

1. Decode (i.e. sample) $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$ from an auxiliary source of random bits
2. Encode $\mathbf{x}$ using $p(\mathbf{x}|\mathbf{z})$
3. Encode $\mathbf{z}$ using $p(\mathbf{z})$

- Expected codelength: $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x},\mathbf{z})\right]$

BB-ANS



- Note the stack behavior of the encoding/decoding
- Arithmetic coding aligns with queues/streams
- ANS provides coding scheme compatible with stacks

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

How well does BB-ANS work?

- Assumptions to investigate:
  - Finite precision approximation of log 1/p
    - Always the case with ANS
  - Inefficiency in encoding the first datapoint
    - Hopefully amortized well
  - VAE has continuous latent variables
    - See next slide
  - Need for “clean” bits
    - See next next slide

[James Townsend, Thomas Bird, David Barber, “*Practical lossless compression with latent variables using bits back coding,*” ICLR 2019]

BB-ANS -- VAE has continuous latent variables

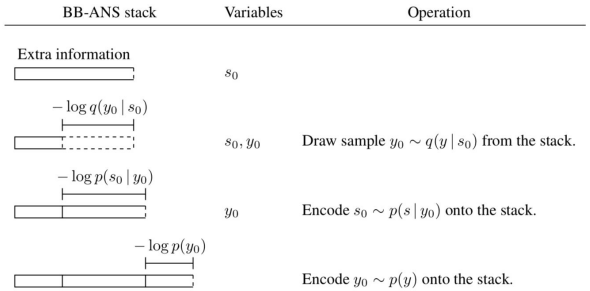
- We can discretize the continuous distribution
- Let's assume we discretize z at level δz
- Expected encoding length with bits-back:

$$-\mathbb{E}_{Q(z_i|x)}\left[\log \frac{p(x|z_i)p(z_i)\delta z}{q(z_i|x)\delta z}\right]$$

$\rightarrow$ δz cancels out, so only cost is in KL between continuous distribution and discrete approximation [Hinton & van Camp, 1993]

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

BB-ANS -- Are the bits clean?



- We assume we sample from $q(z|x)$
- But is our random bit source truly random?
- The bits come from compressing the previous datapoint
- z was encoded with $p(z)$, but came from $q(z|x)$
- What matters is $q(z)$
- And really: $\mathrm{KL}(q(z)||p(z))$
- VAE objective has this as one of the terms, but in practice non-zero

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

How well does VAE bits-back work?

Dataset	Raw data	VAE test ELBO	BB-ANS	bz2	gzip	PNG	WebP
Binarized MNIST	1	0.19	**0.19**	0.25	0.33	0.78	0.44
Full MNIST	8	1.39	**1.41**	1.42	1.64	2.79	2.10

Table 2: Compression rates on the binarized MNIST and full MNIST test sets, using BB-ANS and other benchmark compression schemes, measured in bits per dimension. We also give the negative ELBO value for each trained VAE on the test set.

Dataset	Raw data	BB-ANS with PixelVAE (predicted)	bz2	gzip	PNG	WebP
Binarized MNIST	1	**0.15**	0.25	0.33	0.78	0.44
ImageNet 64×64	8	**3.66**	6.72	6.95	5.71	4.64

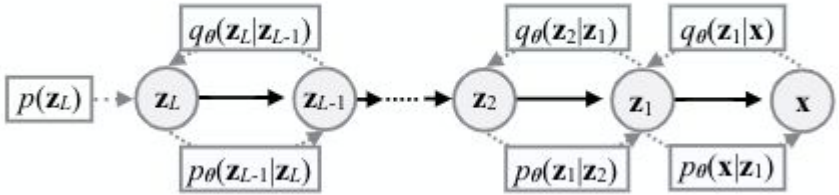
Table 3: Predicted compression of BB-ANS with PixelVAE against other schemes, measured in bits per dimension.

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

Can we do even better?

Quality of encoding depends on quality of ELBO

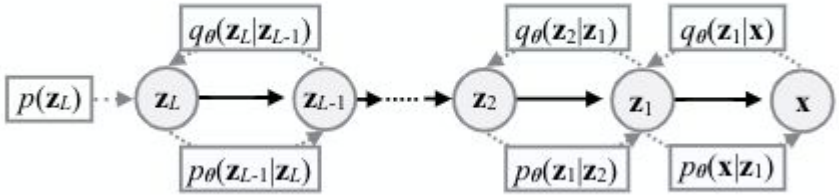
Latent variable models with multiple latent layers tend to achieve better ELBO than with single latent layer



Can we bits-back code for this?

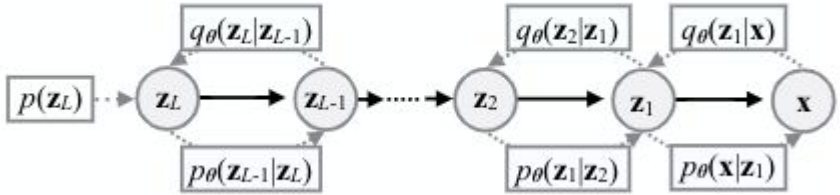
Bit-Swap: Recursive bits-back

- Deep latent variable models can achieve better log likelihoods than single-layer models, especially when encoders and decoders are restricted to be fully factorized. Example model:



Bit-Swap: Recursive bits-back

- Two ways to view this model:
  a. A VAE $p(x) = \sum p(x \mid z_{1:L})\, p(z_{1:L})$
  b. A VAE $p(x) = \sum p(x \mid z_1)\, p(z_1)$, with a VAE prior $p(z_1) = \sum p(z_1 \mid z_2)\, p(z_2)$, with a VAE prior for z_2, etc.



- The two views suggest different bits-back coding algorithms
  a. Apply BB-ANS directly, treating $z_{1:L}$ as a single large latent variable
  b. **Bit-Swap**: Apply BB-ANS to z_1, then **recursively** code z_2, then z_3...

[Kingma F., Abbeel, Ho. ICML 2019]

Bit-Swap: Recursive bits-back

Algorithm 1 BB-ANS for lossless compression with hierarchical latent variables. The operations below show the procedure for encoding a dataset $\mathcal{D}$ onto a bitstream.

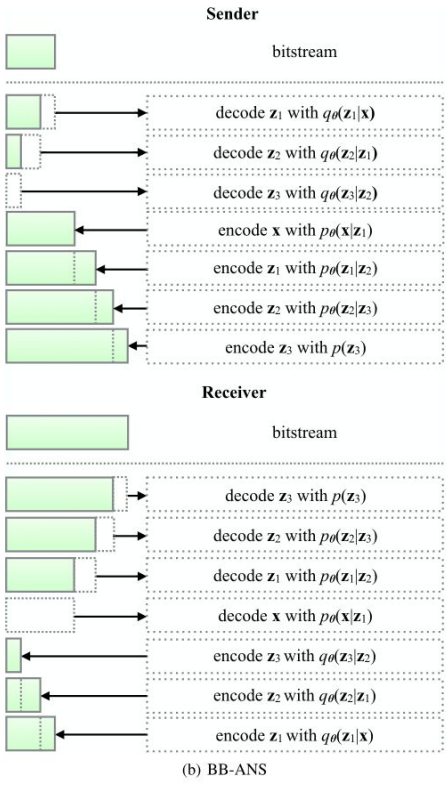
Input: data $\mathcal{D}$, depth L, $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:L})$, $q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})$
Require: ANS
Initialize: bitstream
repeat
  Take $\mathbf{x} \in \mathcal{D}$
  decode $\mathbf{z}_1$ with $q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})$
  for $i = 1$ **to** $L - 1$ **do**
    decode $\mathbf{z}_{i+1}$ with $q_{\boldsymbol{\theta}}(\mathbf{z}_{i+1}|\mathbf{z}_i)$
  end for
  encode $\mathbf{x}$ with $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$
  for $i = 1$ **to** $L - 1$ **do**
    encode $\mathbf{z}_i$ with $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$
  end for
  encode $\mathbf{z}_L$ with $p(\mathbf{z}_L)$
until $\mathcal{D} = \varnothing$
Send: bitstream

Algorithm 2 Bit-Swap (ours) for lossless compression with hierarchical latent variables. The operations below show the procedure for encoding a dataset $\mathcal{D}$ onto a bitstream.

Input: data $\mathcal{D}$, depth L, $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:L})$, $q_{\boldsymbol{\theta}}(\mathbf{z}_{1:L}|\mathbf{x})$
Require: ANS
Initialize: bitstream
repeat
  Take $\mathbf{x} \in \mathcal{D}$
  decode $\mathbf{z}_1$ with $q_{\boldsymbol{\theta}}(\mathbf{z}_1|\mathbf{x})$
  encode $\mathbf{x}$ with $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$
  for $i = 1$ **to** $L - 1$ **do**
    decode $\mathbf{z}_{i+1}$ with $q_{\boldsymbol{\theta}}(\mathbf{z}_{i+1}|\mathbf{z}_i)$
    encode $\mathbf{z}_i$ with $p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{z}_{i+1})$
  end for
  encode $\mathbf{z}_L$ with $p(\mathbf{z}_L)$
until $\mathcal{D} = \varnothing$
Send: bitstream

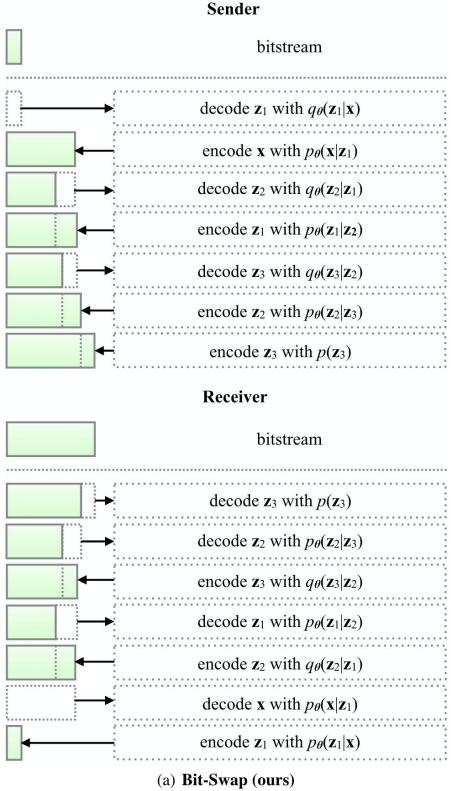
[Kingma F., Abbeel, Ho. ICML 2019]

BB-ANS



(b) BB-ANS

Bit-Swap



(a) **Bit-Swap (ours)**

Bit-Swap Results

Bit-Swap does not need to decode all latents $z_{1:L}$ at once, so it needs fewer auxiliary bits to start bits-back coding.

CIFAR-10 model optimization (columns 2 and 3) and test data compression results (columns 4 to 8) for various depths of the model (column 1).

Depth (L)	ELBO $-\mathcal{L}(\boldsymbol{\theta})$	# Parameters	Avg. Net Bitrate	Scheme	Initial ($n = 1$)	CMA ($n = 50$)	CMA ($n = 100$)
1	4.57	45.3M	-	-	-	-	-
2	3.83	45.0M	3.85 ± 0.77	BB-ANS	12.66 ± 0.61	4.03 ± 0.11	3.93 ± 0.08
				Bit-Swap	6.76 ± 0.63	3.91 ± 0.11	3.87 ± 0.08
4	3.81	44.9M	3.82 ± 0.83	BB-ANS	22.30 ± 0.83	4.19 ± 0.12	4.00 ± 0.09
				Bit-Swap	6.72 ± 0.67	3.89 ± 0.12	3.85 ± 0.09
8	3.78	44.7M	3.79 ± 0.80	BB-ANS	44.24 ± 0.87	4.60 ± 0.12	4.19 ± 0.09
				Bit-Swap	6.53 ± 0.74	3.86 ± 0.12	3.82 ± 0.09

[Kingma F., Abbeel, Ho. ICML 2019]

Bit-Swap Results

Table 1: CIFAR compression results over the course of 100 timesteps (datapoints) for Bit-Swap vs BB-ANS. The cumulative moving average in bits/dim is reported at various timesteps. (There is a discrepancy between bits/dim reported here and the ELBO of the models due to the fact that only hundreds of datapoints are processed – a small fraction of the test set.)

Depth (L)	ELBO $-\mathcal{L}(\boldsymbol{\theta})$	Codelength w/o overhead	Scheme	Initial ($n = 1$)	CMA ($n = 50$)	CMA ($n = 100$)
2	4.59	4.90 ± 0.90	BB-ANS	28.40 ± 0.13	5.31 ± 0.03	5.13 ± 0.05
			Bit-Swap	24.73 ± 0.22	5.24 ± 0.03	5.10 ± 0.04
3	4.39	5.19 ± 1.43	BB-ANS	41.57 ± 0.56	5.87 ± 0.13	5.58 ± 0.10
			Bit-Swap	25.75 ± 0.08	5.52 ± 0.14	5.38 ± 0.10
4	4.45	4.80 ± 0.99	BB-ANS	54.29 ± 0.71	5.76 ± 0.09	5.30 ± 0.06
			Bit-Swap	26.21 ± 0.06	5.18 ± 0.09	5.02 ± 0.09

[Kingma F., Abbeel, Ho. ICML 2019]

Bit-Swap Results

Table 2: MNIST compression results over the course of 100 timesteps (datapoints) for Bit-Swap vs BB-ANS. Similar comments apply as Table 1.

Depth (L)	ELBO $-\mathcal{L}(\boldsymbol{\theta})$	Codelength w/o overhead	Scheme	Initial ($n = 1$)	CMA ($n = 50$)	CMA ($n = 100$)
2	1.24	1.25 ± 0.30	BB-ANS	3.31 ± 0.25	1.27 ± 0.05	1.27 ± 0.03
			Bit-Swap	2.33 ± 0.24	1.25 ± 0.05	1.26 ± 0.04
3	1.25	1.26 ± 0.30	BB-ANS	4.38 ± 0.25	1.31 ± 0.05	1.29 ± 0.04
			Bit-Swap	2.38 ± 0.22	1.27 ± 0.05	1.27 ± 0.04
4	1.34	1.34 ± 0.31	BB-ANS	5.57 ± 0.26	1.41 ± 0.06	1.39 ± 0.04
			Bit-Swap	2.49 ± 0.27	1.35 ± 0.06	1.35 ± 0.04

[Kingma, Abbeel, Ho, 2019 -- paper not yet on arxiv as of 2/20, these are preliminary results to be updated in the future]

Outline

- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
- Theoretical limits
- Coding Considerations
- Autoregressive Models and Arithmetic coding
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- ***Flow Models***
- Lempel-Ziv

Compression with Flows

- Flows are likelihood-based models, so there should be some corresponding compression algorithm

$$-\log p(\mathbf{x}) = -\log p(\mathbf{z}) - \log \left|\det \mathbf{J}(\mathbf{x})\right|$$

$$\mathbf{z} \sim \mathcal{N}(0, I)$$

Coding Continuous Data

- Flows define probability density functions:

$$-\log p(\mathbf{x}) = -\log p(\mathbf{z}) - \log \left|\det \mathbf{J}(\mathbf{x})\right|$$

  - Cannot code continuous data. This needs infinite precision!
- Instead, discretize the data to high precision
  - Tile data space with hypercubes of volume δ_x
  - Let B(x) be the unique hypercube containing x
- And code according to the derived probability mass function:

$$P(\bar{\mathbf{x}}) := \int_{B(\bar{\mathbf{x}})} p(\mathbf{x})\, d\mathbf{x} \qquad -\log P(\bar{\mathbf{x}}) \approx -\log p(\bar{\mathbf{x}})\delta_x$$

- Desired codelength:

$$-\log p(\bar{\mathbf{x}})\delta_x = -\log p(f(\bar{\mathbf{x}})) - \log \left|\det \mathbf{J}(\bar{\mathbf{x}})\right| - \log \delta_x$$

Compression with Flows

- Now we have a probability mass function for a flow:

$$-\log p(\bar{\mathbf{x}})\delta_x = -\log p(f(\bar{\mathbf{x}})) - \log\left|\det \mathbf{J}(\bar{\mathbf{x}})\right| - \log \delta_x$$

- Are we done?
- No. Naive coding takes exponential resources.
- Must harness model structure for computational efficiency
  - We have seen how to do this for AR models and VAEs
  - What about flows?

Compression with Flows

$$-\log p(\bar{\mathbf{x}})\delta_x = -\log p(f(\bar{\mathbf{x}})) - \log\left|\det \mathbf{J}(\bar{\mathbf{x}})\right| - \log \delta_x$$

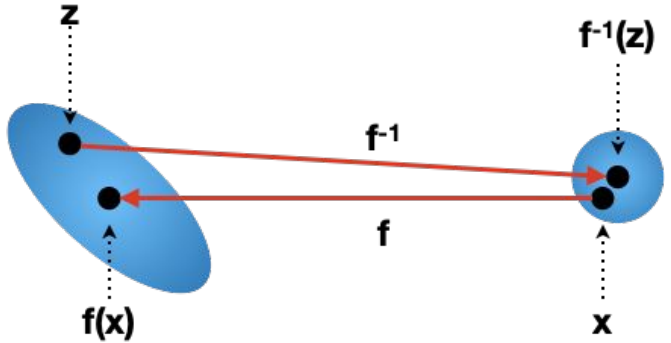
- Naive attempt: code z = f(x) using the prior p(z) (using the same discretization scheme)
  - This does not work. If the flow is trained well, the distribution of latents will match the prior, so there is no compression.
  - To get the Jacobian determinant term, we must account for how the flow changes volume

Compression with Flows

$$-\log p(\bar{\mathbf{x}})\delta_x = -\log p(f(\bar{\mathbf{x}})) - \log\left|\det \mathbf{J}(\bar{\mathbf{x}})\right| - \log \delta_x$$

- Better attempt: construct encoder and decoder and apply bits-back coding

$$\tilde{p}(\mathbf{z}|\mathbf{x}) := \mathcal{N}(\mathbf{z}; f(\mathbf{x}), \sigma^2 \mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^\top) \quad \text{and} \quad \tilde{p}(\mathbf{x}|\mathbf{z}) := \mathcal{N}(\mathbf{x}; f^{-1}(\mathbf{z}), \sigma^2 \mathbf{I})$$



Local Bits-Back Coding

$$\text{Decode } \bar{\mathbf{z}} \sim \mathcal{N}(f(\bar{\mathbf{x}}), \sigma^2 \mathbf{J}(\bar{\mathbf{x}})\mathbf{J}(\bar{\mathbf{x}})^\top)\delta_z$$
$$\text{Encode } \bar{\mathbf{x}} \text{ using } \mathcal{N}(f^{-1}(\bar{\mathbf{z}}), \sigma^2 \mathbf{I})\delta_x$$
$$\text{Encode } \bar{\mathbf{z}} \text{ using } p(\bar{\mathbf{z}})\delta_z$$

Resulting codelength:

$$\mathbb{E}_{\mathbf{z}} L(\mathbf{x}, \mathbf{z}) = -\log p(\mathbf{x})\delta_x + O(\sigma^2)$$

This encoder and decoder defines a VAE whose variational bound matches the flow log likelihood, up to first order.

[Ho, Lohn, Abbeel. NeurIPS 2019]

Local Bits-Back Coding: Algorithms

- Direct implementation requires computing and factorizing the Jacobian
  - $O(d^3)$ time and $O(d^2)$ space
  - Better than exponential, but not good enough for high dimensional data
- Solution: Specialize further
  - Autoregressive flows: code one dimension at a time. Jacobian free, linear time.
  - Composition: code one layer, then recursively code the next layer
  - **RealNVP family: Jacobian free, linear time & parallelizable**

[Ho, Lohn, Abbeel. NeurIPS 2019]

Compression with Flows: Discrete Data

- We achieved the codelength $-\log P(\bar{\mathbf{x}}) \approx -\log p(\bar{\mathbf{x}})\delta_x$
  - Suitable for high precision data
  - But not for low precision data (16-32 bits per dimension extra)
- Solution: apply bits back to the extra precision

$$P(\mathbf{x}) := \int_{[0,1)^d} p(\mathbf{x}+\mathbf{u})\, d\mathbf{u}$$

$$\mathbb{E}_{\mathbf{u}\sim q(\mathbf{u}|\mathbf{x})}\left[-\log \frac{p(\mathbf{x}+\mathbf{u})}{q(\mathbf{u}|\mathbf{x})}\right] \geq -\log \int_{[0,1)^d} p(\mathbf{x}+\mathbf{u})\, d\mathbf{u} = -\log P(\mathbf{x}).$$

- Resulting codelength is the variational dequantization bound

Local Bits-Back Coding

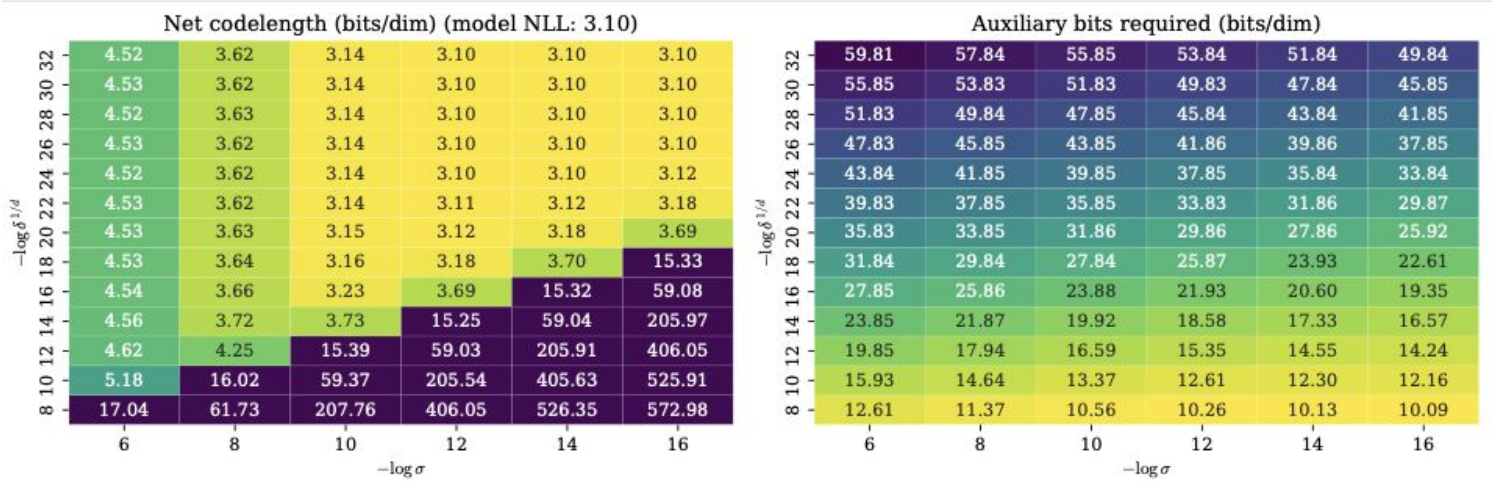
Resuls for Flow++, a RealNVP-type model

Compression algorithm	CIFAR10	ImageNet 32x32	ImageNet 64x64
Theoretical	3.116	3.871	3.701
Local bits-back (ours)	3.118	3.875	3.703

[Ho, Lohn, Abbeel. NeurIPS 2019]

Local Bits-Back Coding

- Caveat: large auxiliary bits requirement
- Need high precision for Jacobian to be an accurate representation of the flow model



Outline

- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
- Theoretical limits
- Coding Considerations
- Autoregressive Models and Arithmetic coding
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- Flow Models
- ***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.

LZ77

- Why does this work? Roughly, for an ergodic source, the average time T to look into the past to see x is 1/p(x)
- The integer T can be coded with approximately log(T) bits
- So after a long time, E[log(1/p(x))] bits per symbol have been used, which is optimal.

Outline

- Compression: What and Why
- Universal lossless compressor?
- Coding of symbols
- Theoretical limits
- Coding Considerations
- Autoregressive Models and Arithmetic coding
- VAE, Bits-Back Coding, Asymmetric Numeral Systems (ANS)
- Flow Models
- Lempel-Ziv

Some Extra Readings

- Townsend et al. HiLLoC: lossless image compression with hierarchical latent variable models (ICLR 2020)
  - Compression on large images using a fully convolutional deep latent variable model
- Havasi et al. Minimal Random Code Learning (ICLR 2019)
  - Alternative to bits-back coding
  - Encoder draws $K=2^{KL(q||p)}$ samples $z_{1:K} \sim p(z)$, then transmits a random number from 1 to K to code one particular z
  - Decoder uses the same random number generator to recover the sample z
  - This is a low-bias sample from q, and takes $\log K = KL(q||p)$ bits to encode z. This achieves the bits-back codelength without bits back.
- Yang et al. Feedback Recurrent Autoencoder (2019)
  - Encoder/decoder architecture for lossy compression for sequential data
  - As time progresses, the encoder simulates the decoder so it can provide more informative bits for the future, which saves on codelength