Lecture notes from the course “Data Compression With Deep Probabilistic Models” by Prof. Robert Bamler, University of Tuebingen, 1 June 2021; more course materials at https://robamler.github.io/teaching/compress21/

Where Were We?

- Information theory & coding theory
  - Source & channel coding
  - Symbol codes (Shannon, Huffman)
  - Theoretical bounds
    - Example: compressing natural language (Problem 3.2)
  - Backwards coding (Problem Set 5)
  - Stream codes
    - Asymmetric Numeral Systems (ANS)
    - Arithmetic Coding & Range Coding

SO FAR

- Probabilistic models & machine learning
  - Probability theory (entropy, random vars)
  - Goal: efficiently capture relevant correlations
    - Autoregressive models
    - Latent variable models & marginal distribs
      - Bayesian inference
      - Scalable approximate Bayesian inference
      - Deep latent variable models (e.g. VAEs)

NEXT STEPS

Neural Compression
Stream Codes

- Reminder: 2 pairs of theorect bounds for lossless comp.
  1. Optimal symbol code:

\[
H_p(X_i) \leq L_{opt} < H_p(X_i) + 1 \text{ bit per symbol}
\]

(expected code word length = expected bit rate per symbol)

2. For any lossless compression:

\[
H_p(X) \leq \mathbb{E}_p[R_p(X)] \leq H_p(X) + 1 \text{ bit per message}
\]

(bit rate of entire message \(X\) with symbol/word codes)

\[\Rightarrow\] Problem: satisfying upper bound is prohibitively expensive 😞

\[\Rightarrow\] In ML-based compression, \(H_p(X_i)\) is often \(< 1\) bit

\[\Rightarrow\] Symbol codes would have high overhead

Toy example: bent coin: \(P(X_i = "\text{heads}") = \alpha \quad (\alpha \in (0,1))\)
\(P(X_i = "\text{tails}") = 1 - \alpha\)

\[\Rightarrow H_p(X_i) = -\alpha \log_2 \alpha - (1 - \alpha) \log_2 (1 - \alpha) =: H_2(\alpha)\]

\[\text{e.g., for } \alpha = 0.1: H_2(\alpha) \approx 0.47\text{ bits}\]

\[\Rightarrow\] Symbol code:

\[>\text{factor of 2 overhead}\]

Goal: alleviate overhead of symbol codes while maintaining \(O(1)\) computational cost \(\Rightarrow\) stream codes

Strategy: amortize bit rate over symbols \(\Rightarrow\) code words
Intuition: message: \( x = (x_1, x_2, x_3) \)

inf. content:
symbol code: (Shannon coding)
stream codes

---

Important Stream Codes

- **Asymmetric Numeral Systems (ANS)**
  - (Duda et al., 2015)

- **Arithmetic Coding**
  - variant: Range Coding

\[ \text{stack} (\text{"last into first"}) \]
\[ \text{queue} (\text{"first into last"}) \]
\[ \text{useful for fixed length coding with latent variable models} \]
\[ \text{useful for autoreg. model} \]
Asymmetric Numeral Systems (ANS)

Exercise 1: Decimal System

Consider the task of compressing a (variable-length) string of statistically independent and uniformly distributed symbols from the alphabet $X = \{0, \ldots, 9\}$. Here, “uniformly distributed” means that all symbols from the alphabet occur with equal probability.

(a) What is the entropy per symbol?

(b) If you were to compress this sequence of symbols with an optimal symbol code, what would be the expected code word length (i.e., the expected bit rate per symbol)?

(c) Can you do better than an optimal symbol code? Describe your approach first in words, then implement it in Python or in pseudo code (about 4 lines of code for encoding and 4 lines of code for decoding).

*Hint:* you may assume that integers in your programming language can be of arbitrary size (as is the case in Python), that they support basic arithmetic operations (addition, subtraction, multiplication, and integer division with remainder) and that they are represented internally as bit strings. In Python 3, integer division is notated as double-slash `//`.

(d) What is the expected bit rate per symbol of your method from part (c), in the limit long messages? Compare your result to the results from parts (a) and (b).

Exercise 2: Ternary System

All parts (a)-(d) exactly as in Exercise 1 above, but this time the alphabet is only of size three. Thus, for each $i \in \{1, \ldots, \kappa\}$, you have a symbol $X_i \in \{0, 1, 2\}$ with probabilities $P(X_i = 0) = P(X_i = 1) = P(X_i = 2) = \frac{1}{3}$.
Observations: Conversion between different positional numeral systems
operate as a stack
> amortizes compression bits over symbols
> optimally compresses a sequence of symbols
if these symbols
(i) all are from same alphabet
(ii) are uniformly distributed over the alphabet
(iii) (Therefore) are statistically independent

→ Let’s remove these constraints 😊

Constraint (i): mixing symbols with different alphabets just works 😊

Constraint (ii): consider non-uniform $P(x_i)$
→ use bits-back trick
→ approximate $P$ by $P_{\text{ANS}}$, which represents all probabilities in fixed-point arithmetic
→ parameter $n \in \mathbb{N}$ (typically a power of 2)

$$P_{\text{ANS}}(x_i = \xi_i) = \frac{m(\xi_i)}{n}$$

→ partitions the range $\xi_1, ..., n-1$ into disjoint subranges for each $x_i \in \mathcal{X}$.

$$\xi a(x_i), ..., a(x_i) + m(x_i) - 1$$
\[ X = \{ \alpha^{\prime}, \beta^\prime, \gamma^\prime \} \]

\[ m(\alpha^\prime) \text{ points} \quad m(\beta^\prime) \text{ points} \quad m(\gamma^\prime) \text{ points} \]

\[ a(\omega) \quad a(\beta^\prime) + m(\beta^\prime) - 1 \quad a(\gamma^\prime) \quad a(\gamma^\prime) + m(\gamma^\prime) - 1 \]

Interpret \( P_{\text{ANS}}(X_i) \) as the marginal dit.

of the latent variable model

\[
P_{\text{ANS}}(X_i; Z) = \frac{P(Z)}{m_{\text{ans}}} P_{\text{ANS}}(X_i; Z)
\]

where \( Z \in \{0, 1, \ldots, n-1\} \)

with uniform prior, i.e., \( P(Z = z) = \frac{1}{n} \) \forall z

and likelihood

\[
P(X_i = x_i | Z = z) = \begin{cases} 1 & \text{if } z \in \{a(x_i), \ldots, a(x_i) + m(x_i) - 1\} \\ 0 & \text{else} \end{cases}
\]

Claim: \( P_{\text{ANS}}(X_i = x_i) = \frac{1}{m(x_i)} \sum_{Z=0}^{n-1} P_{\text{ANS}}(X_i = x_i, Z = z) \)

⇒ Apply bit-by-bit coding:

1) Find posterior density.

\[
P(Z = z_i | X_i = x_i) = \frac{P(Z = z_i) P(X_i = x_i, Z = z_i)}{\sum_{Z=0}^{n-1} P(Z = z_i) P(X_i = x_i, Z = z_i)}
\]

\[
= \begin{cases} \frac{1}{m(x_i)} & \text{if } z_i \in \{a(x_i), \ldots, a(x_i) + m(x_i) - 1\} \\ 0 & \text{else} \end{cases}
\]
Thus, both prior & posterior are uniform distributions, just with different alphabet sizes.

\[ \Rightarrow \text{Bits-back coding:} \]

\[ \text{\underline{encode}}(x_i): \]

\[ \bullet \text{\underline{decode}} \ z_i \text{ using } P(Z_i | X_i = x_i) \]

\[ \text{uniform over } \{z_i(1), \ldots, z_i(n), \ldots, z_i(m)-1\} \]

\[ \bullet \text{\underline{encode}} \ x_i \text{ using } P(X_i | Z_i = z_i) \]

\[ = \log_2 X_i(1) + \log_2 X_i(2) + \cdots + \log_2 X_i(n) \]

\[ \Rightarrow \inf \text{\underline{constant}} = 0 \]

\[ \bullet \text{\underline{encode}} \ z_i \text{ using } P(z_i) \]

\[ \text{uniform over } \{z_i(1), \ldots, n-1\} \]

\[ \Rightarrow \text{net bit rate for 1 symbol} \]

\[ -\log_2 m(x_i) + \log_2 n = -\log_2 \frac{m(x_i)}{n} \]

\[ \Rightarrow \text{\underline{decoder}:} \]

\[ \bullet \text{\underline{decode}} \ z_i \text{ using } P(Z_i) \text{ identify } x_i \]

\[ \bullet \text{\underline{encode}} \ z_i \text{ using } P(Z_i | X_i = x_i) \]

\[ \text{Constraint (iii): statistical independence of encoded symbols} \]

\[ \Rightarrow \text{now that we can encode each symbol with its individual probabilistic model (see experiment in video recording) we can model correlations using either a latent variable model (with another layer of bits-back coding) or an autoregressive model (as we've done before).} \]
(Naive) implementation in Python:

```python
class AnsCoder:
    def __init__(self, precision):
        self.precision = precision
        self.mask = (1 << precision) - 1
        self.compressed = 1

    def encode(self, symbol, scaled_probabilities):
        z = self.compressed & scaled_probabilities[symbol]
        self.compressed //= scaled_probabilities[symbol]
        for prob in scaled_probabilities[:symbol]:
            z += prob
            self.compressed = (self.compressed << self.precision) + z

    def decode(self, scaled_probabilities):
        z = self.compressed & self.mask
        self.compressed >>= self.precision
        for i, prob in enumerate(scaled_probabilities):
            if prob > z:
                symbol = i
                break
            else:
                z -= prob
        self.compressed = (self.compressed * scaled_probabilities[symbol] + z)

return symbol
```

Improving Run-Time Cost of ANS (to $O(k)$)

> metaphor: money

- savings account
- bulk transfers of round amounts
- spending account (or cash)
- odd incomes
- smallish odd payments (e.g., $\leq 17.39$)
similar for (streaming) ANS:

"bulk"

"head"

- $O(1)$ cost per symbol
- $O(k)$ cost per message

Implement on Problem Set 7

On Problem Set 6: Optimality of positional numerical system code