II Concepts

- **Lossless compression**
  - data = sequence of symbols, from a known predefined alphabet (discrete setting)
  - set of symbols
    \[ S = \{ a_0, a_1, a_2, \ldots, a_n \} \]
    \[ S' = \{ 0, 1, 2, 3, \ldots \} \]
    \[ S'' = \{ a', b', c', \ldots, e' \} \]
  - encoder: data \( \rightarrow \) binary string
    - injective
    - otherwise: not decodable
    - uniquely decodable code
  - decoder: binary string \( \rightarrow \) data
    - if encoder not surjective: not optimal compression
  - lossless \( \iff \) decoder \( \circ \) encoder = \( \text{Id} \)

- **Prediction**
  - given a sequence of observed variables: \( x_1, x_2, \ldots, x_n \), predict \( x_{n+1} \)
  - i.e., give the probability distribution of each possible signal
    \[ p(x_{n+1} | x_1, \ldots, x_n) \]
  - predictor: function: data \( (x_1, \ldots, x_n) \) \( \rightarrow \) probability of next variable
    \[ \text{base: vector:} \begin{pmatrix} p(x_1 | x_2, \ldots, x_n) \\ p(x_2 | x_1, \ldots, x_n) \\ \vdots \\ p(x_{n-1} | x_1, \ldots, x_{n-2}) \end{pmatrix} \]
    \[ \Sigma = 1 \]

- **Generative model**
  - model able to generate new data
  - data \( D \) is generated according to its model probability \( p(D) \)
    \[ \text{as often as} \]
    \[ \text{2 points} \rightarrow \text{distribution over data} \]
    \[ \text{sample from it} \]

II From one concept to the other

**Prediction \( \Rightarrow \) Generation** [easy]
- start from \( x_1 \) \( \sim \) \( S' \)
  - or taken from a learned distribution
- choose \( x_{n+1} \) according to the probability law \( p(x_{n+1} | x_1, \ldots, x_n) \)
  - used to predict next symbol
- iterate
  - generate data this way
- \( p(D) = \prod p(x_1 | x_2, \ldots, x_{i-1}) \)
  \[ (x_1, \ldots, x_n) \]
  \[ = p(x_1) p(x_2 | x_1) \ldots p(x_n | x_1, x_2) \]
  \[ = p(D) \]
  - Bays theorem applied \( n \) times
Generative \rightarrow \text{Prediction} \quad \text{[by sampling \ldots might be slow]}

- given \( x_1 \ldots x_n \), how to estimate the probability distribution of \( x_{n+2} \)?
- generate new data \( D \) many many times, according to the generative law
- check how frequent each \((x_1 \ldots x_n, x_{n+2})\) is generated \( \rightarrow p(x_1 \ldots x_n) \)
- compute \( p(x_{n+1} \mid x_1 \ldots x_n) = p(x_1 \ldots x_n) / p(x_1 \ldots x_n) \) \quad \text{Bayes}

Compression \rightarrow \text{Prediction}
- idea: strings whose encoding is shorter are more probable
- \( L_0 \text{ character with } L = -\log p \rightarrow p = 2^{-L} \)
- complete a string:
  \[ L(\text{string} + c') \rightarrow p(c') \propto 2^{-L(\text{string} + c')} \]
- given \( x_1 \ldots x_n \), build all possible completed sequences \( x_1 \ldots x_n, x_{n+2} \)
- sequence \( S_k : x_1 \ldots x_n, k \text{ by symbol} \)
- compress each of them independently, and check their encoding length
  \( L_k = \text{size} (\text{encode} (S_k)) \)
- define \( p(x) = 2^{-X} \)
- \( \sum_k p_k \) needs to be a prob dist: \( \sum_k p_k(x) = 1 \)
  \( p(x) \) normalize \( p(x) \) by a global factor if necessary
- \( L_0 \text{ normalized } (p(x)) \) by a global factor if necessary
- \( L_0 \), we'll see later that the norm is already \( 1 \) if the character is optimal

Predictor \rightarrow \text{Compression} \quad \text{[the main point of this lesson]}
- lesson from examples: if we have good prior information on the distribution \( x_{n+2} \mid x_1 \ldots x_n \)
  then we don't need too much information to communicate to know which symbol is chosen
  for \( x_{n+2} \)
- encode just that information \( \rightarrow \) so that it's decodable later
- entropy \( = \text{average length of that code} \)
  \[ H[p(x_{n+2} \mid x_1 \ldots x_n)] = \sum_{x_{n+2}} p(x_{n+2} \mid x_1 \ldots x_n) \log p(x_{n+2} \mid x_1 \ldots x_n) \]
- can we prove that bound?
- \( \log \) is it possible to reach that bound? how?

III. Encoders
- issue: when to stop reading data

\* self-delimiting code
  \* \text{to code on integer } n \quad \text{(not bounded)} \quad n \in \mathbb{N}

- need \( b = \lceil \log_2 n \rceil \)
- learn to tell the length of the sequence to encode
- encode: \( \log_2 \text{ bits} = \log_2 n \)
or stop at first iteration and encode differently

\[ \left\lceil \log n \right\rceil + \left\lceil \log \log n \right\rceil \times 2 \]

001 111 1001000

Since \( 72 \) \( \in \mathbb{N} \)

0000...01

padded with 0

Then \( n^{-1} \) '0' followed by '1'

* Codes with "out-of-file" decoder

- postfix: read the data, until you meet a special code meaning "end"
- To do this: make binary blocks of fixed size and use one of them for "end"

\[
\begin{align*}
1100 & 0110 1010 0010 0000 \\
& 0100 1010 1011 0010 0000
\end{align*}
\]

- obvious cost: \(-\log_2(\text{"end")}) = 4 + 4 bits
- hidden cost: can't use symbols forming "end" at other places
- cost: number of blocks \( \times -\log_2(1\text{-pld}) \)

4-bit block: \( 2^4 = 16 \) possible patterns

4-5 for end

45 remains

* Prefix codes

when the code of a string cannot be the beginning of the code of another string

\[
\begin{align*}
10011 & 10110001 \\
& 23
\end{align*}
\]

David Mackay's book pg 104 for examples

(such codes can be written as binary trees)

As can be shown later, that without losing generality, one can assume that a code is prefix (otherwise, rebuild another code with same coding lengths)
Bounds, optimal codes and Kraft inequality

- Entropy $H$ makes use of symbol probabilities (to produce shorter average encoding length)
  - More frequent symbols should have shorter encoding lengths
- Hint: $2^{-\text{length of symbol}}$ should be $\propto p(s)$
  - So expect $\text{Length (Code of } s) = -\log(p(s)) \propto H$ of entropy

Kraft inequality
- Given an encoder, let's denote by $(l_i)$ the list of the lengths of all possible codewords
  - Then the code is uniquely decodable $\Rightarrow \sum_i 2^{l_i} \leq 1$
  - And reciprocally:
    - If $(l_1)$ satisfies $\sum_i 2^{l_i} < 1$, then there exists an encoder (prefix-code) whose codewords have this length distribution
- Moreover: If equality $\sum_i 2^{l_i} = 1$, the code is said to be "complete"

Math proof:
- Note $S = \sum_i 2^{l_i}$

  - Define $S^n = (\sum_i 2^{l_i})^n = \sum_i 2^{nl_i}$
    - Count terms $\sim$ draw it using the number of ways $\binom{n}{l_i}$
    - $S^n \leq \binom{n}{\max l_i}$
      - Chain $l_i$ together

  - Proof by drawing:
    - List all the codewords
      - 0000, 0101, 1010
      - Not possible to have too many short codewords
      - $4 \times 1 \times 1 \times 1 \times 1 = 4$
Lower bound on compression length

\[ \text{Length (encoding } X) \geq H(X) \]  

with equality only if the code length is coherent with the probability \( P(i) \) in the code \( \Sigma i \geq \log p(i) \)

Proof: Gibbs inequality + Kraft

\[ \mathbb{E} \left[ \ell(x) \right] = \sum_{i \in X} p(i) \ell_i \]

with distribution over symbols of the alphabet

\[ = \sum_{i \in X} p(i) (-\log S - \log q_i) \]

\[ = -\log S \sum_{i \in X} p(i) \log q_i \]

\[ q_i = \frac{2^{\ell_i}}{S} \]

\[ S q_i = 2^{\ell_i} \]

\[ \ell_i = -\log S - \log q_i \]

\[ \geq H(p) \]

Upper bound

It's possible to build a code such that:

Source coding theorem:

There exists a variable-length encoding \( C \) of an ensemble \( X \) such that the average length of an encoded symbol, \( L(C, x) \), satisfies:

\[ L(C, x) \leq H(X) + 2 \]

Proof: Kraft with \( \ell = \log p_i \)

Constructive proof: Huffman coding (best possible with integer nb of bits/symbol)

(non-integer nb of bits)

Reach \( H(p) \) rate
Huffman coding

1) write down one node per symbol with its probability
2) iteratively, join the 2 nodes with lowest probability, and form a new node with price = sum of the 2
3) this forms a binary tree
4) code = position in the tree

Optimality proof:
- Frequency x node encoding length
  1) For a given set of node encoding lengths, optimise set of codeword lengths
  2) Suppose another prefix code is better with a set of codeword lengths
     For last symbol, symbols are ≠

Arithmetic coding

A = { A, B, C, D } alphabet

Let with prefix order: A, B, C, D... -

Length associated to signal 'B' is its frequency

Draw a random number uniformly in [0,1]
Let it have R% of chances to be in the 88m
Let us pick randomly symbols with the frequency of the distribution

Real: the probability that a string starts with "BB" is the width of the bin coding for "BB"

1) Encoding
   Given string: DBAC -> yields an interval
   [0.5] -> choose the binary representation of a real number (the falls into the bin, i.e. x ∈ [0.5])

2) Decoding
   Pick randomly (uniformly) a real number 0.5
   and decode it (its binary representation)
→ when to stop decoding (real number in binary format)

```
A(p) = \begin{cases} 
\ldots & \text{for EOF} \\
1 & \text{for special symbols} \\
0 & \text{for symbol A} 
\end{cases} 
```

\[ A(p) = \begin{cases} 
0 & \text{for symbol A} \\
1 & \text{for special symbols} \\
\ldots & \text{for EOF} 
\end{cases} \]

\[ p(A) = (1-e)p(A) \quad p(\text{EOF}) = e \]

\[ H(p) = -e \log_e (1-e) \log_e (1-e) + (1-e) H(p) \]

→ choose ε: try the best for your data

\[ 2 \times H(p) \]

By for EOF (End of File)

Alphabet = \{ (latin characters + ' + punctuation marks) \}

"Hello world!" → "Hello world! EOF"

Huffman coding is an approximation of arithmetic coding with the constraint that bin sizes should be powers of 2 (i.e. \(2^n\))

Additional remarks

Law over integers

Case 1: encoding length for number \( n \):

\[ p(n) \propto 2^{-\log_2 n} = \frac{1}{n \log_2 n} \]

\( \sum p(n) \) does converge \( \Rightarrow \) can be normalized as a probability distribution

Case 2: encoding length:

\[ p(n) \propto \frac{1}{n \log_2 \log_2 n} \]

\( \sum_{n} p(n) \) converges \( \Rightarrow \) Kraft says yes

Case 3: \( \frac{1}{n} \) or \( \frac{1}{n \log_2 \log_2 n} \) ? \( \Rightarrow \) not possible because \( \sum p(n) \to \infty \)

Example: two cheaters → fill the forms with false numbers
Let's don't know information theory... they pick the first digit randomly uniform in \( [1,2,3] \)

\[ p(3) = \frac{1}{3} \]

\( p(y) \) vs. uniform

\[ p(y) \sim \frac{1}{\log(y)} \]

\[ p(y) \sim \frac{1}{y} \]

\[ p(100,000) \approx p(800,000) \]

\[ y \text{ is about } 10 \text{ times more probable than} \]

\[ y \]

Set of 3 digit numbers

Next generation: notion of typicality

data (made with \( n \) symbols) is typical of a model if its average coding length / symbol is close to the entropy of the model:

\[ \mathbb{E}[(x - \mu)^2] = \sigma^2 \text{ standard deviation} \]

Please model efficiency or error in bits

\( \rightarrow \) fit a model to a ML task \( \rightarrow \) encoder

\[ p \rightarrow \neg p \]

\( \neg \neg p \)

---

**Ex**: test compression/prediction/generation with Markov chains

- of practical sense
  
  - IID model
  
  - Markov chains with various orders

- compression:

  \[ \text{ht_long} \rightarrow \text{ht_short} \text{.net} \]

  \[ \text{ht_long} \text{.net} \rightarrow \text{ht_short} \text{.net} \]

  \[ \text{~\textbf{if~able~to~compress~file~by}~1\%~more} \]

\[ \text{~\textbf{by~wikipedia}} \]

Conclusion:

- good model for data: good for every task: prediction, generation, compression

- measure model suitability in bits

- see generative models as compressors

- baseline: \( \log_2 \) (Zipf1.2n)

- \( \log_2 \) is continuous with non-integer number of bits, thanks to arithmetic coding

\[ \log \]