Problem Set 2

published: 27 April 2021
discussion: 3 May 2021

Data Compression With Deep Probabilistic Models
Prof. Robert Bamler, University of Tuebingen

Course material available at https://robamler.github.io/teaching/compress21/

Problem 2.1: Kraft-McMillan Theorem

In the lecture, we stated and partially proved the Kraft-McMillan Theorem:

Theorem (Kraft-McMillan)**.** *The following two statements are true:*

(a) All B-ary uniquely decodable symbol codes C satisfy the Kraft inequality,

$$\sum_{x\in\mathfrak{X}} \frac{1}{B^{\ell(x)}} \leq 1. \tag{1}$$

Here, $\mathfrak{X}$ is the alphabet of the symbol code and $\ell(x)$ is the length $|C(x)|$ of the code word $C(x)$ (i.e., the number of B-ary bits that make up $C(x)$).

(b) For all functions $\ell : \mathfrak{X} \to \{0, 1, 2, \ldots\}$ that satisfy the Kraft-inequality Eq. 1, there exists a B-ary prefix-free symbol code (i.e., a B-ary prefix code) C with code word lengths ℓ, i.e., $|C(x)| = \ell(x)\ \forall x \in \mathfrak{X}$.

Answer the following questions about the Kraft-McMillan theorem:

(a) Prove the following statement by combining both parts (a) and (b):

For every uniquely decodable symbol code C, there exists a prefix code $\tilde{C}$ with the same code word lengths, i.e., $|\tilde{C}(x)| = |C(x)|\ \forall x \in \mathfrak{X}$.

In the lecture, we proved part (a) of the Kraft-McMillan theorem but we didn't complete the proof of part (b). Let's do this now. Consider Algorithm 1 on the next page, which we introduced in the lecture.

(b) Line 4 of Algorithm 1 claims that $\xi \in [0, 1)$. Why is this always the case at this point in the algorithm?

(c) Denote the value of ξ *after* the update in Line 3 as ξ_x (where x is the iteration variable of the `for` loop). Now consider two symbols $x, x' \in \mathfrak{X}$ with $x \neq x'$ and, without loss of generality, $\xi_{x'} > \xi_x$. Argue that $\xi_{x'} \geq \xi_x + B^{-\ell(x)}$. Then argue that neither can $C(x)$ be a prefix of $C(x')$, nor can $C(x')$ be a prefix of $C(x)$.

(d) *(Advanced:)* Algorithm 1 is limited to a finite alphabet $\mathfrak{X}$ because the `for`-loop would not terminate if $\mathfrak{X}$ was infinite. How could you use the same idea to prove part (b) of the Kraft-McMillan Theorem in the case of a counably infinite alphabet?

Algorithm 1: Constructive proof of Kraft-McMillan theorem part (b).

```
Input:  Base $B \in \{2, 3, \ldots\}$, finite alphabet $\mathfrak{X}$,
        function $\ell : \mathfrak{X} \to \{0, 1, 2, \ldots\}$ that satisfies Eq. 1.
Output: Code book $C : \mathfrak{X} \to \{0, \ldots, B-1\}^*$ of a prefix code that
        satisfies $|C(x)| = \ell(x)\ \forall x \in \mathfrak{X}$.
1 Initialize $\xi \leftarrow 1$;
2 for $x$ in $\mathfrak{X}$ in order of nonincreasing $\ell(x)$ do
3     Update $\xi \leftarrow \xi - B^{-\ell(x)}$;
4     Write out $\xi \in [0, 1)$ in its $B$-ary expansion: $\xi = (0.???\ldots)_B$;
5     Set $C(x)$ to the first $\ell(x)$ bits following "0." in the above $B$-ary expansion
        of $\xi$ (pad with trailing zeros to length $\ell(x)$ if necessary);
```

Problem 2.2: Entropy

In the lecture, we (re-)introduced the entropy to base B of a probability distribution p:

$$H_B[p] = \mathbb{E}[-\log_B p(x)] = -\sum_{x \in \mathfrak{X}} p(x) \log_B p(x). \tag{2}$$

(a) In the literature, the subscript B will often be dropped. Depending on context, entropies are understood to be either to base 2 (mostly in the compression literature) or to the natural base e (in mathematics, statistics, or machine learning literature). How do $H_2[p]$ and $H_e[p]$ relate to each other?

(b) Consider a tuple of $m \in \mathbb{N}$ symbols. The symbols are i.i.d. (statistically independent and identically distributed), i.e., the probability $\tilde{p}$ of a tuple of m symbols is $\tilde{p}\big((x_1, \ldots, x_m)\big) = \prod_{i=1}^m p(x_i)$. Show that

$$H_B[\tilde{p}] = m H_B[p] \qquad \forall m \in \mathbb{N},\ \forall B \in \mathbb{R}_{\geq 0} \tag{3}$$

Problem 2.3: Block Codes and Theoretical Lower Bound for Lossless Compression

In the lecture, we showed that the expected code word length L of a uniquely decodable *symbol code* is lower bounded by the entropy H_B of the symbols. We further showed that this lower bound is nontrivial: for any probability distribution p of symbols, there exists a symbol code (the so-called Shannon Code) that is prefix-free (and therefore uniquely decodable) and for which L comes within less than one bit of this lower bound. Thus, in summary, for the *minimally possible* expected code word length L_{min}, we have

$$H_B[p] \leq L_{\text{min}} \leq L_{\text{Shannon}} < H_B[p] + 1. \tag{4}$$

So far, these results are limited to symbol codes, i.e., codes for which the encoding $C^*(\mathbf{x})$ of a sequence $\mathbf{x} = (x_i)_{i=1}^k$ of symbols x_i is given by simple concatenation of individual context-independent code words $C(x_i)$. In this problem, you will derive a lower bound that holds for *all* lossless B-ary codes, not just for symbol codes. To do so, we introduce the idea of a *Block Code*: Let $m \in \{2, 3, \ldots\}$ and assume that you only care about messages $\mathbf{x} \in \mathfrak{X}^k$ whose length k is a multiple of m, i.e., $k = nm$ for some integer n. You can then group the symbols in the message $\mathbf{x}$ into n blocks of m consecutive symbols each, and construct a symbol code for these blocks.

For example, a message $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$ of length $k = 6$ can be reinterpreted as a message of $n = 2$ blocks of size $m = 3$ each: $\tilde{\mathbf{x}} = \big((x_1, x_2, x_3), (x_4, x_5, x_6)\big)$. In this representation, each block is an element of the product alphabet $\mathfrak{X}^m = \mathfrak{X} \times \mathfrak{X} \times \ldots \times \mathfrak{X}$. One can now construct a code book $\tilde{C}^{(m)} : \mathfrak{X}^m \to \{0, \ldots, B-1\}^*$ for this product alphabet. In particular, we will consider the Shannon Code $\tilde{C}^{(m)}_{\text{Shannon}}$ that one obtains from applying the Shannon Coding algorithm to the product probability distribution $\tilde{p}\big((x_1, \ldots, x_m)\big) = \prod_{i=1}^m p(x_i)$.

(a) Use Eqs. 3 and 4 to derive a lower and an upper bound for the expected length of the encoding *per original symbol (from $\mathfrak{X}$)* for $\tilde{C}^{(m)}_{\text{Shannon}}$. You should find that the lower bound does not change compared to Eq. 4, but the upper bound shrinks, i.e., the range narrows.

(b) In the introduction to this problem, we highlighted the restrictions of symbol codes in comparison to arbitrary lossless compression codes: in a symbol code, $C^*(\mathbf{x})$ has to be the concatenation of individual context-independent code words. Now consider a block code for the special case $m = k$. What are its limitations compared to an arbitrary uniquely decodable B-ary lossless compression code that is defined on sequences of k i.i.d. symbols?

(c) What can you say about the Shannon block code $\tilde{C}^{(m)}_{\text{Shannon}}$ with $m = k$ in the (practically relevant) limit of large k? Think about (i) its overhead in expected bits per (original) code word over the theoretical lower bound; and about (ii) the run-time complexity of the Shannon coding algorithm (Algorithm 1) on such a block code as a function of k.

Problem 2.4: Huffman Coding

In the last tutorial, we introduced the Huffman Coding algorithm. It is restated in a more formal manner in Algorithm Box 2. Like Shannon Coding, the Huffman Coding algorithm takes a probability distribution p over symbols, and it constructs a code book for a prefix code C_{Huffman} whose expected code word length L_{Huffman} satisfies the upper and lower bounds from Eq. 4. Unlike Shannon Coding, a code book constructed by Huffman coding is *optimal* in the sense that there is no uniquely decodable symbol code with expected code word length shorter than L_{Huffman}.

Algorithm 2: Huffman Coding (for base $B = 2$).

```
Input:  finite alphabet $\mathfrak{X} = \{1, \ldots, |\mathfrak{X}|\}$, probability distribution $p$ on $\mathfrak{X}$.
Output: Code book $C_{\text{Huffman}} : \mathfrak{X} \to \{0, \ldots, B-1\}^*$ of an optimal prefix code on $\mathfrak{X}$.
1  Initialize a set of tree roots $R \leftarrow \{(p(x), x) : x \in \mathfrak{X}\}$;
2  Initialize a forest $(V, E)$ whose vertices are initialized as $V \leftarrow R$, and with a (so
     far) empty set of edges: $E \leftarrow \emptyset$;
3  Initialize an integer variable $y^* \leftarrow |\mathfrak{X}|$;
4  while $|R| > 1$ do
5      Let $(w, y)$, $(w', y')$ be the two smallest elements of $R$, by lexicographic order;
6      Remove $(w, y)$ and $(w', y')$ from $R$ (but not from $V$);
7      Update $y^* \leftarrow y^* + 1$;
8      Add the new element $\gamma := (w + w', y^*)$ to both $R$ and $V$;
9      Add labeled edges $(\gamma, (w, y), \texttt{label} = 0)$ and $(\gamma, (w', y'), \texttt{label} = 1)$ to $E$;
10 Interpret the resulting tree as a trie of the code book $C_{\text{Huffman}}$: for all $x \in \mathfrak{X}$,
     the code word $C_{\text{Huffman}}(x)$ is obtained by identifying the unique leaf node
     $(w, y) \in V$ with $y = x$, walking along the unique path from the root node to
     said leaf node, and concatenating the labels along the edges of this path.
```

In this problem, we set $B = 2$. (*Note:* to generalize Huffman coding to $B > 2$, you would have to pad the alphabet $\mathfrak{X}$ to a size that is a multiple of $(B-1)$ by "making up" additional symbols with zero probability; we won't concern ourselves with this additional complication here.)

(a) Consider the alphabet $\mathfrak{X} = \{1, 2, 3, 4, 5\}$ and the probabilistic model $p(1) = 0.3$, $p(2) = 0.28$, $p(3) = 0.12$, $p(4) = 0.1$, $p(5) = 0.2$. Execute the Huffman Coding algorithm manually on paper and write down a table for C_{Huffman}. Verify that you obtained a prefix code.

(b) Extend your table from part (a) with a Shannon Code C_{Shannon} for the same model p. To obtain C_{Shannon}, use Algorithm 1 with $\ell(x) := \lceil -\log_2 p(x) \rceil\ \forall x \in \mathfrak{X}$. Verify again that you obtained a prefix code.

(c) Evaluate the entropy $H_2[p]$ and the expected code word lengths L_{Huffman} and L_{Shannon}. Verify that both codes satisfy Eq. 4 and that $L_{\text{Huffman}} \leq L_{\text{Shannon}}$ (for this particular example, you should find that L_{Huffman} is strictly smaller than L_{Shannon}, but equality is possible but for other probability distributions).

(d) Implement the Huffman Coding algorithm in Python by filling in the blanks in the accompanying Jupyter notebook. You may want to refer back to the code examples for Problem 1.3 on last week's problem set.

Problem 2.5: RIP, Simplified Game of Monopoly

Starting next week, we will discuss methods from probabilistic machine learning that will allow us to model complex data sources, such as natural language, images, or videos. We will see that many of the concepts that we illustrated so far with the help of toy-ish probabilistic models (like the “Simplified Game of Monopoly”) will carry over to these more powerful and more realistic deep probabilistic models.

But before we put the Simplified Game of Monopoly to rest, here’s an off-topic brain teaser for you: how would you actually build a 3-sided die as a physical object? What (3-dimensional) shape would it have?

Don’t forget to provide anonymous feedback to this problem set in the corresponding poll on moodle.