CS 229r Information Theory in Computer Science Feb 12, 2019

Lecture 5

Instructor: Madhu Sudan *Scribe: Pranay Tankala*

1 Overview

A universal compression algorithm is a single compression algorithm applicable to input sequences from an entire class of information sources, rather than a specific probability distribution. We will be particularly interested in the class of *Markovian* sources, formally defined in the previous lecture. Today's goal is to work toward a proof that the Lempel-Ziv algorithm is a universal compression algorithm for the class of Markovian sources.

1.1 Logistics

Problem Set 1 is currently being graded; expect feedback early next week. Problem Set 2 will be released soon.

1.2 Schedule

Universal Compression

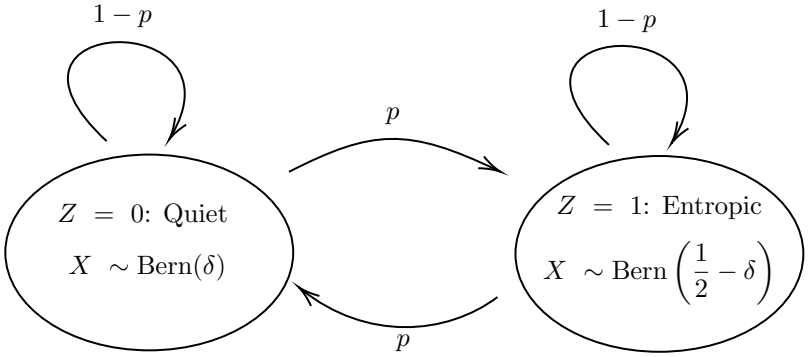
- Hidden Markov Models (HMM)
- A Universal Compressor
- Lempel-Ziv is Universal

2 Hidden Markov Models

We begin with some illustrative examples of hidden Markov models, which are closely related to Markov chains:

2.1 Outer Space

Consider a satellite listening for signals from outer space. Most of the time, there is hardly any information transmission, but occasionally, some highly entropic planetary activity is observed. We model the information source as a hidden Markov model, with two internal states and some small parameters $\delta \to 0$ and $p \to 0$:



We denote the successive states of this information source as $Z_1, Z_2, \ldots, Z_n$, and the bit sequence produced as $X_1, X_2, \ldots, X_n$. We have $X_i \sim \text{Bern}(\delta)$ if $Z_i = 0$ and $X_i \sim \text{Bern}\left(\frac{1}{2} - \delta\right)$ if $Z_i = 1$. Our goal is to compress the sequence $X_1, \ldots, X_n$.

2.2 Shannon's n-gram Model

In Shannon's foundational paper, he discusses a Markov chain to model English, in which letters are drawn randomly according to a probability distribution conditional on the previous $n-1$ letters. Here is Shannon's example of a sequence produced using trigram frequencies:

```
IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES
OF THE REPTAGIN IS REGOACTIONA OF CRE.
```

http://math.harvard.edu/∼ctm/home/text/others/shannon/entropy/entropy.pdf

2.3 General Markovian Sources

We now provide a general definition of hidden Markov models that abstracts the key features of the previous two examples. Intuitively, a hidden Markov model consists of a Markov chain of "hidden" states $Z_1, Z_2, \ldots$, that can only be observed through a sequence of related variables $X_1, X_2, \ldots$ (which do not necessarily form a Markov chain). More formally,

Definition 1. A *hidden Markov model* is characterized by:

- a Markov chain $M \in \mathbb{R}^{k \times k}$, where

$$M_{ij} = \Pr[\text{next state is } j | \text{current state is } i].$$

- k distributions $P^{(1)}, \ldots, P^{(k)}$ supported on Ω, with $P^{(i)}$ the distribution of X given $Z = i$.

When discussing a Markov chain M, we will always assume that M is irreducible and aperiodic, so that M has a unique stationary distribution $\Pi \in \mathbb{R}^k$ satisfying $\Pi M = \Pi$. We will also assume that $Z_1 \sim \Pi$, which implies that $Z_i \sim \Pi$ for every i.

Definition 2. The *entropy rate* of a Hidden markov model is $\lim_{n\to\infty} H(X_n | X_1, \ldots, X_{n-1})$.

Under our assumptions on M, the above limit always exists. Indeed, we have

$$\begin{aligned} H(X_n | X_1, \ldots, X_{n-1}) &\leq H(X_n | X_2, \ldots, X_{n-1}) && \text{(conditioning reduces entropy)} \\ &= H(X_{n-1} | X_1, \ldots, X_{n-2}) && \text{(we've assumed a stationary process),} \end{aligned}$$

so the values $H(X_n | X_1, \ldots, X_{n-1})$ are monotonically decreasing, and hence converge to a limit $H(M)$ from above.

Remark In example 2.1.1, we have $k = 2$, $M = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$ and distributions $\text{Bern}(\delta)$ and $\text{Bern}\left(\frac{1}{2} - \delta\right)$ for the quit and entropic states, respectively. However, we lack a closed form expression for the entropy rate $H(M)$ in terms of p and δ. In fact, the best approximation algorithms we have at our disposal to compute $H(M)$ are still quite coarse.

Remark Without our assumptions on M, there would be no reason to expect the values $H(X_n | X_1, \ldots, X_{n-1})$ to converge to a limit. For example, if we were to take any Markov chain M and place an unavoidable "intermediate" node along each edge, the entropy $H(X_n | X_1, \ldots, X_{n-1})$ could be wildly different for even and odd n.

3 A Universal Compressor

In order to begin to describe universal compression algorithms for Markovian sources, we need

Theorem 3 (Asymptotic Equiparition Property for Markovian Sources)**.** *For every hidden Markov model $(M, P^{(1)}, \ldots, P^{(k)})$ and $\varepsilon > 0$, for sufficiently large ℓ, there exists a set $S \subseteq \Omega^\ell$ such that for every start state Z_1,*

1. $\Pr[(x_1, \ldots, x_\ell) \notin S] \leq \varepsilon$
2. *For all $(\alpha_1, \ldots, \alpha_\ell) \in S$,*

$$\frac{1}{|S|^{1+\varepsilon}} \leq \Pr[(x_1, \ldots, x_\ell) = (\alpha_1, \ldots, \alpha_\ell)] \leq \frac{1}{|S|^{1-\varepsilon}}.$$

3. $|S| \approx 2^{(H(M)\pm\varepsilon)\ell}$

The set S is called the typical set*.*

Note that the AEP for Markovian sources is more general than the version of the AEP we saw before, which dealt only with i.i.d. sequences of random variables.

Exercise 4. *Using the AEP, as stated above, argue that*

$$-\frac{1}{\ell} \log p(X_0, \ldots, X_{\ell-1}) \to H(M)$$

in probability as $\ell \to \infty$.

3.1 A Non-Universal Compressor

We'll first use the AEP to describe a remarkably simple compression algorithm for a *known* Markovian source M. Suppose we wish to encode a string $x_1 \cdots x_n$ produced by M. Take ℓ and S as in the statement of the AEP, and suppose that $n \gg \ell$. We first divide the string into blocks of length ℓ:

$$\underbrace{x_1 \cdots x_\ell}_{B_1} \underbrace{x_{\ell+1} \cdots x_{2\ell}}_{B_2} \cdots \underbrace{\cdots x_n}_{B_{n/\ell}}.$$

To encode a block B_i, we first use a single bit to indicate whether or not $B_i \in S$. Depending on whether or not $B_i \in S$, the rest of the encoding consists of either $\log_2 |S|$ or $\log_2 |\Omega^\ell|$ bits, respectively. By the AEP, the probability that $B_i \notin S$ is at most ε. Thus, the expected length of a single block's encoding is at most

$$(1 + \log_2 |S|) + \varepsilon \cdot (1 + \log_2 |\Omega^\ell|) \approx (H(M) + \varepsilon + \varepsilon \log_2 |\Omega^\ell|) \cdot \ell,$$

which is very close to the optimal rate of $H(M)$ bits per input symbol. To encode x, we simply concatenate the encodings of $B_1, \ldots, B_{n/\ell}$.

3.2 Universal Huffman Algorithm

The previous algorithm made use of the values ℓ, S. This is problematic for two reasons. First, it limits the algorithm to a *single* Markovian source. Second, we generally do not know how to compute ℓ or S. We attempt to circumvent these issues with what we will call the *universal Huffman algorithm*. Suppose we are given $x_1, \ldots, x_n$ to encode, and a parameter ℓ. The algorithm proceeds in stages:

1. Divide the string into size ℓ blocks as before:

$$\underbrace{x_1 \cdots x_\ell}_{B_1} \underbrace{x_{\ell+1} \cdots x_{2\ell}}_{B_2} \cdots \underbrace{\cdots x_n}_{B_{n/\ell}},$$

   and compute the *empirical frequency*

$$f_w = \text{frequency of } w \text{ among } B_1, \ldots, B_{n/\ell}$$

   for every $w \in \Omega^\ell$.

2. Compute the Huffman code tree for the frequencies $\{f_w\}_{w \in \Omega^\ell}$, and use this to compress the sequence $B_1, \ldots, B_{n/\ell}$.

3. Finally, transmit any encoding of the list of frequencies $\{f_w\}$ computed in part 1, followed by the string $\text{Huffman}(B_1, \ldots, B_{n/\ell}, \{f_w\}_w)$ computed in part 2.

Exercise 5. *Prove that for all Markovian sources M and $\varepsilon > 0$, as ℓ and n grow, we have*

$$\Pr[|\textit{Univ-Huffman}(x_1, \ldots, x_n)| > n \cdot (H(M) + \varepsilon)] \to 0$$

Remark We don't know much about how large n must be, as a function of ℓ, in order for the above algorithm to consistently perform well. We know even less about how large ℓ must be, as a function of the underlying Markov chain. If you were surprised by the seemingly inexpensive computations needed for the universal Huffman algorithm, be warned that the real complexity is hidden in the size of n, which may need to be exponentially large for the compression to work well.

Exercise 6. *Prove, using principles of complexity theory, that the total compression time for a universal compression algorithm must be at least exponential in k, the number of states of the Markov chain.*

4 Lempel-Ziv is Universal

We begin with a review of the Lempel-Ziv algorithm to compress a string $x = x_1 \cdots x_n$, $x_i \in \Omega$.

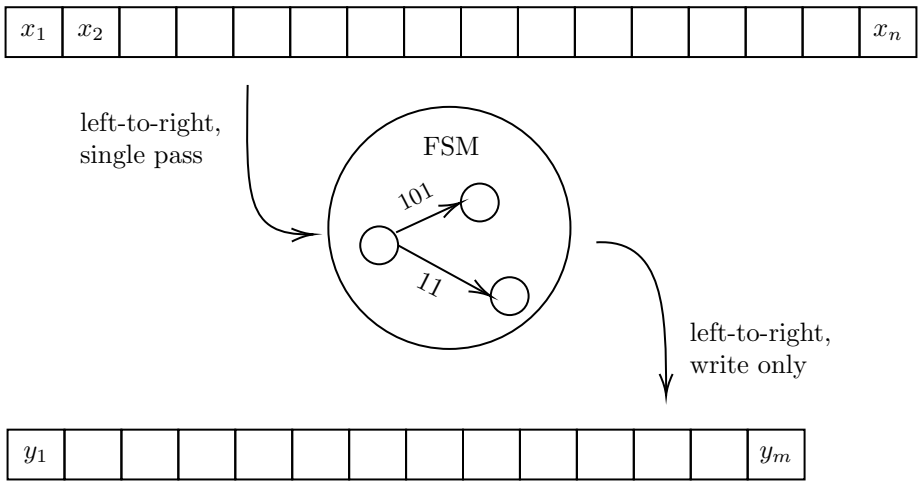
1. First, we parse x into consecutive phrases $s_0 \circ s_1 \circ s_2 \circ \cdots \circ s_t$, such that
   (a) s_0 is the empty string.
   (b) $s_i = s_{j_i} \circ b_i$, for some $b_i \in \Omega$ and $j_i < i$.
   (c) $s_i \neq s_j$ for $i \neq j$.
2. Encode each j_i using $\log t + o(\log t)$ bits, and encode each b_i using $\log |\Omega|$ bits.
3. Transmit the encoding of each j_i and b_i.

The resulting compression length is $(t \log t)(1 + o(1))$ in total.

4.1 Analyzing Lempel-Ziv

Before analyzing the Lempel-Ziv algorithm, we need to introduce one more important construct: the finite state compressor.

Definition 7. A *finite state compressor* (FSC) is a finite state machine (FSM) that writes 0 or more symbols to an output tape during each state transition (see diagram).



Finite state compressors have a very important property related to Markovian sources, which will be crucial in our analysis of Lempel-Ziv:

Theorem 8. *For any hidden Markov model M and $\varepsilon > 0$, there exists a finite state compressor that achieves a compression rate of $(1+\varepsilon)H(M)$.*

Remark It turns out that the proof of the theorem involves constructing a FSC that reads its input sequence in blocks of size ℓ. As a result, the number of states S in the machine exceeds Ω^ℓ, which is exorbitantly large compared to the k-state machine M used to produce the input!

Recall that we want to prove that Lempel-Ziv achieves a nearly optimal compression rate. By the previous theorem, it suffices to show that the performance of Lempel-Ziv is "not much worse" than that of any finite state compressor. To make this comparison precise, we define $C(x)$, for a string $x \in \Omega^*$, as the maximum integer t such x can be written as the concatenation of t distinct strings $Y_1, \ldots, Y_t$ in Ω^*. One might interpret $C(x)$ as a measure of the inherent complexity of compressing the string x. Using $C(x)$, we can state the following important bounds:

1. In the Lempel-Ziv parsing $x = s_1 \circ \cdots \circ s_t$, we have $t \leq C(x)$.
2. The length of the Lempel-Ziv encoding of x satisfies
$$|LZ(x_1, \ldots, x_n)| \leq t \log t(1 + o(1)) \leq C(x) \log C(x)(1 + o(1)).$$
3. For a string x,
$$|x| \geq C(x) \log \frac{C(x)}{4|\Omega|^2}$$
4. Finally, for any S-state compressor F, the compression length satisfies
$$|\mathrm{Comp}_F(x)| \geq C(x) \log \frac{C(x)}{4|\Omega|^2 S^2}.$$

In the next lecture, we will further discuss the inequalities above, which provide the relationship between Lempel-Ziv and FSCs needed to complete our analysis.

Addendum inserted by Madhu

The proofs of the three inequalities claimed above is not hard nor is it crucial for future lectures, so instead of using half a lecture to go over them, let me just add them here and conclude why this implies that the Lempel-Ziv algorithm is a univeral compressor. We start with the second one above, and then turn to the first and finally the last.

Claim 9. *For every string $x \in \Sigma^*$,*

$$|x| \geq (1 - o(1))C(x) \log_{|\Sigma|} C(x).$$

Proof. Let $x = y_1 \circ y_2 \cdots y_t$ with y_i's being distinct elements of Σ^*. To minimize the length of x we might as well assume that the y_i's are lexicographically ordered strings with the shortest ones first. So there are $|\Sigma|$ strings of length 1, $|\Sigma|^2$ strings of length 2 etc., till we get to t distinct strings. The claim now follows from some calculations (omitted here). □

Claim 10. *The length of the Lempel-Ziv encoding of $x = (x_1, \ldots, x_n)$, denoted $|LZ(x)|$, satisfies*

$$|LZ(x)| \leq t \log t(1 + o(1)) \leq C(x) \log C(x)(1 + o(1)),$$

where the $o(1)$ term goes to zero as $n \to \infty$.

Proof. This follows from the fact that Lempel-Ziv produces a split of $x = y_1 \cdots y_t$ with distinct y_i's. It follows that $t \leq C(x)$ and since each y_i is encoded using at most $logt \cdot (1 + o_t(1))$ bits, it follows that the compression length

$$|LZ(x)| \leq t \log t(1 + o_t(1)) \leq C(x) \log C(x)(1 + o_t(1)).$$

It is left as an exercise to see that $o_t(1) = o_n(1)$ for Lempel-Ziv (i.e., that t grows as n grows. □

Claim 11. *For every S-state compressor F, the compression length satisfies*

$$|Comp_F(x)| \geq (1 - o(1))C(x) \log \frac{C(x)}{|S|^2} = (1 - o(1))C(x) \log C(x).$$

Proof. Fix a partition $x = y_1 \cdots y_t$ of x with y_i's being distinct. For states a, b of F, let S_{ab} be the set of indices $i \in [t]$ such that the compressor is in state a at the beginning of parsing y_i and ends in state b after parsing y_i. Let $y'_i \in \{0,1\}^*$ denote the string output by F while parsing y_i. The crux of this proof is the claim that $y'_i \neq y'_j$ for $i, j \in S_{a,b}$, i.e., if the compressor starts and ends in the same state then it must produce distinct outputs on distinct inputs (or else the "compression" is not invertible). Once we have this, we have that $\sum_{i \in S_{a,b}} |y'_i| \geq (1 - o(1))|S_{a,b}| \log_2 |S_{a,b}|$ (from the first claim above). And thus $|Comp_F(x)| = \sum_{a,b} \sum_{i \in S_{a,b}} |y'_i| \geq (1 - o(1)) \sum_{a,b} |S_{a,b}| \log_2 |S_{a,b}|$. Finally by the Cauchy-Schwartz inequality and using the fact that $\sum_{a,b} |S_{a,b}| = t$ we get that $|Comp_F(x)| \geq (1 - o(1)) \sum_{a,b} |S_{a,b}| \log_2 |S_{a,b}| \geq (1 - o(1))t \log(t/|S|^2)$. □

Putting the last two claims together we have that Lempel-Ziv compresses almost as well as an finite state compressor. Combining with Theorem 8 we have the Lempel-Ziv achieves a compression length of at most $(1 + \varepsilon)H(M)$ per input symbol.