In J. A. Storer and M. Cohn (editors), *Proceedings of the Data Compression Conference 1996*, IEEE Computer Society Press, Los Alamitos, CA.

Free energy coding

Brendan J. Frey and Geoffrey E. Hinton
Department of Electrical Engineering
University of Toronto
10 King's College Road
Toronto, Canada, M5S 1A4

E-mail: frey@cs.toronto.edu and hinton@cs.toronto.edu
Phone: +1-416-978-7391
Fax: +1-416-978-1455

May 28, 1996

Abstract

In this paper, we introduce a new approach to the problem of optimal compression when a source code produces multiple codewords for a given symbol. It may seem that the most sensible codeword to use in this case is the shortest one. However, in the proposed *free energy* approach, random codeword selection yields an *effective* codeword length that can be less than the shortest codeword length. If the random choices are Boltzmann distributed, the effective length is optimal for the given source code. The expectation-maximization parameter estimation algorithms minimize this effective codeword length. We illustrate the performance of free energy coding on a simple problem where a compression factor of two is gained by using the new method.

Index Terms — source coding, data compression, bits-back, free energy, expectation maximization, Boltzmann distribution.

Introduction

Ordinary source codes map each input symbol to a unique codeword. This mapping is determined by an underlying source model whose parameters may be fixed by hand, estimated using a data set, or estimated on-line. As source models grow in sophistication, we expect to see cases where the model that is best suited to the data maps each input symbol to more than one codeword. For example, consider a source that outputs real numbers that are distributed according to a mixture of two Gaussians. These numbers are truncated to some precision to form a set of symbols. The component distributions and the output distribution are shown in figure 1a, where the truncation effect is left out for the sake of graphical

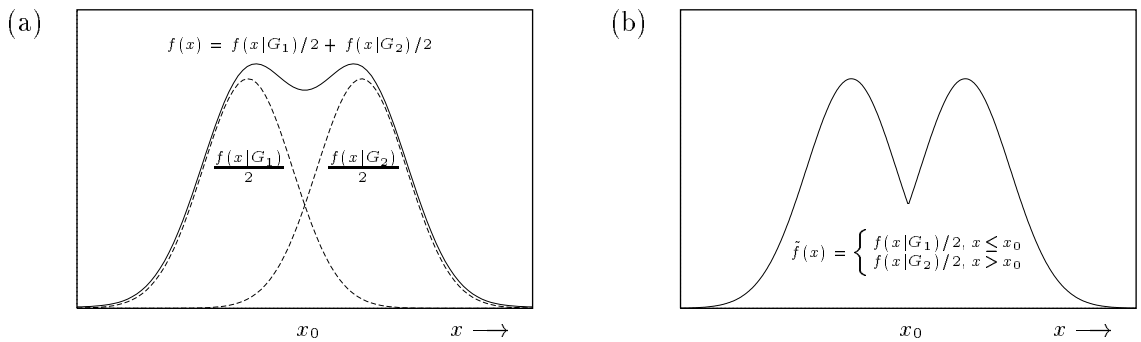

Figure 1: The most natural source model may produce multiple codewords for a given symbol. (a) shows a source with a single binary hidden variable which identifies from which Gaussian, G_1 or G_2, the symbol value x is sampled. Values of x near x_0 are likely to have come from either Gaussian. (b) shows the resulting coding density effectively used if we were to always pick the shorter codeword. This density wastes coding space because it is wrongly shaped and has an area significantly less than unity.

simplicity. The most natural source model to use in this case is one that requires one bit to specify from which Gaussian a given symbol was produced plus however many bits are needed to code the symbol using that Gaussian. However, the identity of the Gaussian that produced a given symbol is often ambiguous — in particular, a number near x_0 is likely to have come from either Gaussian. In these cases the source model maps each symbol to two codewords: one for each Gaussian. If we were to always pick the shorter of the two codewords, we would effectively be assuming the symbols were distributed as in figure 1b. However, this distribution is obviously incorrect — it is not even normalized — and will lead to suboptimal compression. One obvious way around this problem is to compile a table of symbol values and their corresponding marginal densities computed by summing the probability densities of the two Gaussians. Codewords of optimal length can then be produced using this table. However, this approach suffers from two disadvantages. Firstly, we must tabulate the probabilities for every symbol — a task that is computationally intractable and memory intensive for high-dimensional data. Secondly, we must sum over all possible configurations of the *hidden variables* (in this case the identity of the Gaussian) — a task that is computationally intractable for non-local models since the number of terms in the sum grows exponentially with the number of hidden variables. We are left, then, with the challenge of trying to achieve optimal compression when a source code produces multiple codewords for a given symbol.

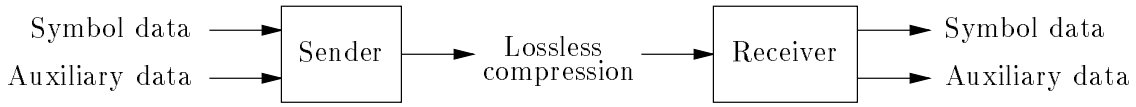

Figure 2: A scheme in which auxiliary data is communicated along with the primary symbol data in order to achieve optimal compression when the source code produces multiple codewords for a given symbol.

Bits-back

The approach we take to solve this problem is motivated by the "bits-back" argument of Hinton and Zemel [4]. It turns out that by randomly selecting codewords, we can communicate extra *auxiliary* information along with the data we are coding. The bits communicated in the auxiliary data will more than make up for the excess codeword lengths that result from not always using the shortest codeword. This method of coding is shown in figure 2. Suppose in the Gaussian mixture example that a sender wishes to code a truncated value that is twice as likely under G_1 as it is under G_2, and that 2 bits are required to code the truncated value under G_1. Including the single bit required to specify which Gaussian is being used, the two codewords will have lengths $l_1 = 3$ bits and $l_2 = 4$ bits. If the sender always picks the shorter codeword, the average codeword length is 3 bits.

Suppose instead that for this particular truncated value the sender randomly chooses between each of the two codewords equally often (in general, the ratio of choices will depend on the truncated value). It appears the average codeword length in this case is $0.5l_1+0.5l_2 = 3.5$ bits, which is higher than that obtained by always choosing the shorter codeword. However, this cost is effectively lowered because the receiver can recover information from the random choice in the following manner. Say the sender has well-compressed auxiliary data available in the form of a queued bit stream with "0" and "1" having equal frequency. When coding, the sender uses the next bit in the auxiliary data queue to choose between G_1 and G_2. The sender then produces a codeword that will have an average length of 3.5 bits. When decoding, the receiver reads off the bit that says which Gaussian was used and then determines the truncated value from the codeword. Given the decoded value, the receiver can run the same encoding algorithm as the sender used, and determine that a choice of equal probability was made between G_1 and G_2. Since the receiver also knows which Gaussian was selected, the receiver can recover the queued auxiliary data bit that was used to make the choice. In this way, on average 1 bit of the auxiliary data is communicated at no extra cost. These recovered bits are called *bits-back*. If the auxiliary data can be made useful, the average effective codeword length is reduced by 1 bit due to the savings, giving an effective

average length of 2.5 bits; less than the 3 bits required by the shortest codeword.

This procedure can be formalized by defining a distribution that is used to select codewords for a given symbol, x:

$$Q(y|x), \tag{1}$$

where y indexes the possible codewords for the given symbol. Letting $l(x, y)$ be the length of codeword y for a specific x, the average codeword length is

$$\mathcal{E}(x) \equiv \sum_y Q(y|x) l(x, y). \tag{2}$$

The average bits-back is the information content (entropy) of the distribution used to select codewords:

$$\mathcal{H}(x) \equiv -\sum_y Q(y|x) \log_2 Q(y|x). \tag{3}$$

The difference between equations 2 and 3 gives the effective average codeword length, $\mathcal{F}(x)$:

$$\mathcal{F}(x) \equiv \mathcal{E}(x) - \mathcal{H}(x). \tag{4}$$

Since this quantity is analogous to the variational *Helmholtz free energy* from statistical physics [9], we refer to the effective average codeword length as the *free energy*. Free energy coding makes gains over shortest codeword selection by taking into account the existence of multiple codewords of similar length. The codeword selection distribution which minimizes free energy is the Boltzmann distribution, given by

$$Q^*(y|x) \equiv \frac{2^{-l(x,y)}}{\sum_{\hat{y}} 2^{-l(x,\hat{y})}}. \tag{5}$$

We denote by "*" those quantities determined from the Boltzmann distribution. This distribution gives the optimal effective codeword length for the given source code. In the above example:

$$Q^*(G_1|x) = 2^{-3}/(2^{-3} + 2^{-4}) = 2/3; \tag{6}$$

$$Q^*(G_2|x) = 2^{-4}/(2^{-3} + 2^{-4}) = 1/3; \tag{7}$$

$$\mathcal{E}^*(x) = \frac{2}{3}(3 \text{ bits}) + \frac{1}{3}(4 \text{ bits}) = 3.333 \text{ bits}; \tag{8}$$

$$\mathcal{H}^*(x) = -\frac{2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3}) = 0.918 \text{ bits}; \tag{9}$$

$$\mathcal{F}^*(x) = 3.333 \text{ bits} - 0.918 \text{ bits} = 2.415 \text{ bits}. \tag{10}$$

This is the minimum free energy for the given example. A slightly higher than optimal free energy of 2.5 bits was obtained above using $Q'(G_1|x) = Q'(G_2|x) = 0.5$.

For complex source models, the computation of the denominator in equation 5 is sometimes intractable; in these cases, it is impossible to obtain the exact Boltzmann distribution.

However, various methods exist for approximating the Boltzmann distribution. The free energy given by such an approximation can be compared to the minimum free energy given by the Boltzmann distribution:

$$\mathcal{F}(x) - \mathcal{F}^*(x) = \sum_y Q(y|x) \log_2 \frac{Q(y|x)}{Q^*(y|x)}. \tag{11}$$

This is the Kullback-Leibler number (*aka.* relative entropy) between the candidate codeword selection distribution and the Boltzmann distribution. It is always positive and will yield the number of extraneous bits in the compression as a result of not using the Boltzmann distribution. Selecting the shortest codeword uses a number of extraneous bits given by

$$\mathcal{F}^{\text{short}}(x) - \mathcal{F}^*(x) = \sum_y Q^{\text{short}}(y|x) \log_2 \frac{Q^{\text{short}}(y|x)}{Q^*(y|x)} = -\log_2 Q^*(y^{\text{short}}|x). \tag{12}$$

If many codewords have roughly equal lengths, $Q^*(y^{\text{short}}|x)$ is significantly less than unity indicating that picking the shortest codeword is far from optimal.

Free energy coding

To implement the communication scheme shown in figure 2, we need a general method of achieving bits-back. In the example given above, there were two codewords and each was selected equally often so that a single bit could be used for bits-back. If the codeword selection distribution is dyadic[1], Huffman coding [5] may be used. Here, we consider the case of an arbitrary codeword selection distribution. In this more general case, it is not so easy to see how random codeword choices can be made without losing auxiliary data information. To address this problem, consider the operation of an arithmetic decoder [8,11]. It receives a finite-precision number on $[0, 1)$ and extracts from it a series of decisions according to a table of probabilities. If a collection of uniformly distributed finite-precision numbers on $[0, 1)$ is decoded, we will obtain a collection of decisions whose distribution exactly matches the table of probabilities. Figure 3 shows how an arithmetic decoder can be used to conserve the information in the auxiliary data when making random codeword choices. The probabilities associated with the decisions form the table of the arithmetic decoder, while the auxiliary data defines a random number to be "decoded". (It is important that the auxiliary data have a roughly uniform distribution.) Each column of the figure corresponds to a single codeword choice and is partitioned into several possible outcomes with the height of each partition equal to the probability of the corresponding outcome. It is easy to see that if the random number defined by the auxiliary data is uniform, codeword choices will be made according to the codeword selection distribution (as shown for a particular case by the shaded partitions). It is also easy to see that by applying an arithmetic encoder to the sequence of decisions, we can regenerate the random number.

[1] *ie.*, each probability is an integral power of 2.

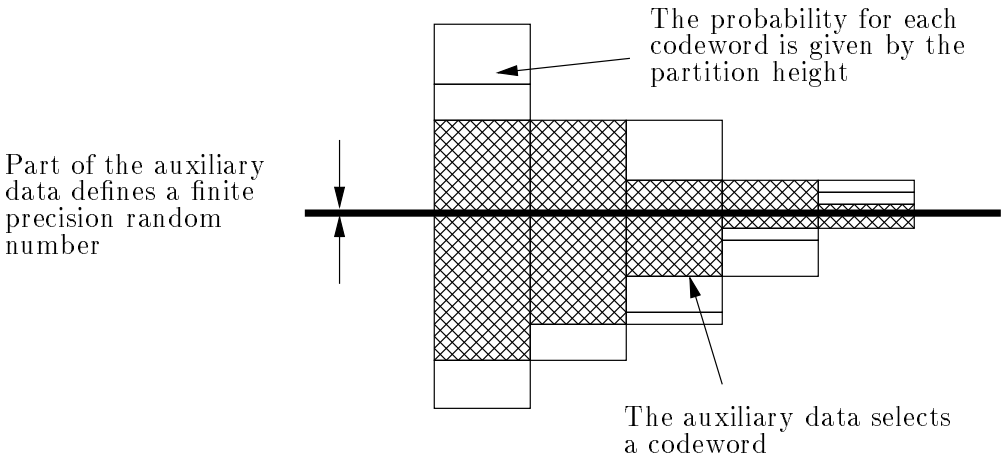

Figure 3: Feeding a random number into an arithmetic decoder with appropriate probabilities (shown by the partition heights within a column) selects codewords (shaded partitions), while at the same time conserving information.

With useful auxiliary data, this scheme empowers us to communicate bits-back, obtaining bits-back coding. With a good codeword selection distribution, we can achieve optimal effective compression for the given source code. Note that this coding algorithm can be used with a source model that is adaptive and/or has memory. Also, the codewords themselves need not be of integer length, since the source model may simply provide a distribution over codewords and then an arithmetic encoder/decoder pair can be used to communicate the codewords.

Expectation maximization minimizes free energy coding cost

In general, a source model contains a set of parameters, θ, that must be fixed by hand, estimated from a data set, or estimated on-line. Estimating these parameters is a difficult task when there are hidden variables. For a given symbol, it is necessary to determine how the hidden variables combine to give the density for the symbol under the current model. Only then can the parameters be adjusted to increase or decrease the probability of the symbol. There are many algorithms for accomplishing this incomplete data task: the expectation maximization (EM) algorithm [2], the generalized EM algorithms [6], the Baum-Welch algorithm [1], the expectation-conditional maximization algorithms [6], and the incremental EM algorithm [7]. The Baum-Welch algorithm, for example, is known

for its application in training hidden Markov models [3]. Neal and Hinton [7] show that all of these algorithms can be viewed as variants of the EM algorithm that attempt to optimize a single cost function in different ways. The cost function is the free energy given in equation 4, averaged over the data set. Since this is also the average code length for free energy coding, it follows that variants of the EM algorithm estimate model parameters that minimize the free energy coding communication cost.

Experimental results on a simple problem

There are two main reasons for presenting experimental results. Firstly, we wish to show that by using an EM variant to fit a model to data, the resulting source code produces multiple codewords of approximately the same length for a given symbol; *ie.*, the free energy resulting from always choosing the shortest codeword is significantly greater than that resulting from choosing codewords according to the Boltzmann distribution. Secondly, the intricate implementation details demand implementation results. In this section, we present a simple source model that is used to produce a large data set by randomly selecting model parameters and then simulating data. Note that since we have the true underlying source model, we can compute the entropy of the data source. Then, model parameters are estimated using the data set and a variant of the EM algorithm. After discussing the theoretical free energy coding savings, we present results when this model is used in software-implemented versions of free energy coding and, for comparison, coding by shortest codeword selection.

Because of its popularity, the hidden Markov model (HMM) is a good choice as a simple example of free energy coding. In fact, it is simple enough that the hidden variables (state) can tractably be integrated out; however, for the purpose of illustrating how free energy coding works, let us ignore this and assume that integrating out the state is not an option. In its most basic form, the HMM is a probabilistic model of discrete-time sequences. Details of HMM operation and parameter estimation are not given here; a short, clear description of HMM basics is given by Ferguson [3]. It is sufficient to note that the model parameters can be estimated from a data set using the Baum-Welch algorithm (a variant of the EM algorithm) and also that for a given symbol sequence the model can be used to produce a codeword.

The HMM used in our experiments was inspired by the idea of modelling the sequential structure within English words. It has 20 hidden states and can output one of 26 letters ("a" to "z") at a time. Each string is defined as a 10-letter word, so there are 26^{10} possible strings in all. So that we can compare our results with the entropy of the data source, we do not estimate the model parameters using actual English words. Instead, we randomize the parameters of the HMM and then repeatedly simulate the model to obtain a data set consisting of 10,000 strings of 10 letters each. The initial state probabilities are sampled from a gamma distribution ($\alpha = 0.4$, $\beta = 1.0$) and then normalized. The state transition table is determined by randomly picking an entry in each row, setting it to 0.962, and then

Table 1: Compression performances for data set.

	File size (KBytes)	Compression factor
Original 10,000-string ascii file	110.0	100.0%
Shortest codeword selection	25.8	23.5%
Free energy coding	14.7	13.4%
Source entropy	11.7	10.6%

setting the remaining entries to 0.002. The letter output table is set in the same fashion, except 0.975 and 0.001 are used instead of 0.962 and 0.002. Using a method described in [3] we find the entropy of this data source is 9.32 bits/string.

To estimate the parameters of a HMM with the same architecture as above, we first initialize the probabilities to be uniform plus 2% peak-to-peak uniform noise (to break symmetry). After estimating the HMM parameters using 200 iterations of the Baum-Welch algorithm, we find that based on the data set of 10,000 strings, the average shortest codeword length is 21.27 bits/string[2]. Whereas in the model used to generate the data, the average shortest codeword length is similar to the source entropy, in the estimated model it is much higher because it has many different ways of producing the same string. The free energy is 11.80 bits/string, indicating that the model is not optimally fit — an optimal fit would give a free energy equal to the entropy of the source. However, if this codeword length can be achieved in practice, a gain of nearly a factor of two will be made over shortest codeword selection.

We compare an implementation of a coding algorithm that always chooses the shortest codeword with an implementation of a free energy coding algorithm. The former system uses a single arithmetic encoder in the sender and a single arithmetic decoder in the receiver. The latter uses an arithmetic decoder for bits-back and an arithmetic encoder for the codeword in the sender; and an arithmetic decoder for the codeword and an arithmetic encoder for bits-back in the receiver. A binary data file with uniformly random bits is used for auxiliary data. The file sizes and compression factors (percentages of original ascii file) are given in table 1. The compression factor for the free energy coding algorithm comes close to the optimum value given by the source entropy. The compression factor obtained by shortest codeword selection is nearly twice that of the free energy coding value, indicating that a significant practical savings can be made by using the proposed approach.

Conclusions

We have introduced the concept of optimal compression using bits-back. Further, it is apparent that a large body of popular methods for fitting models with hidden variables — the EM variants — produce source models for which shortest codeword selection is

[2]To find the shortest codeword for a given string, we use the Viterbi algorithm [10].

suboptimal in compression efficiency whereas free energy coding is optimal. We have shown that a practical free energy coding algorithm can be implemented and that it does indeed outperform a shortest codeword selection algorithm in compression performance.

Acknowledgements

We thank Radford Neal for extensive, helpful discussions and Glenn Gulak for useful comments. We are grateful to the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre for financial support.

References

[1] Baum, L. E., and Eagon, J. A. 1967. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. AMS* **73**, 360-363.

[2] Dempster, A. P., Laird, N. M., and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B* **39**, 1-38.

[3] Ferguson, J. D. 1980. Hidden Markov analysis: An introduction. In *Proceedings of the Symposium on the Applications of Hidden Markov Models to Text and Speech*, IDA-CRD, Princeton, NJ.

[4] Hinton, G. E., and Zemel, R. S. 1994. Autoencoders, minimum description length, and Helmholtz free energy. In J. K. Cowan, G. Tesauro and J. Alspector (editors), *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Francisco, CA.

[5] Huffman, D. A. 1952. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers* **40**, 1098-1101.

[6] Meng, X. L., and Rubin, D. B. 1992. Recent extensions of the EM algorithm (with discussion). In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith (editors), *Bayesian Statistics 4*, Clarendon Press, Oxford England.

[7] Neal, R. M., and Hinton, G. E. 1996. A new view of the EM algorithm that justifies incremental and other variants. Submitted to *Biometrika*.

[8] Rissanen, J., and Langdon, G. G. 1976. Arithmetic coding. *IBM Journal of Research and Development* **23**, 149-162.

[9] Thompson, C. J. 1988. *Classical Equilibrium Statistical Mechanics*. Clarendon Press, Oxford England.

[10] Viterbi, A. J., and Omura, J. K. 1979. *Principles of Digital Communication and Coding*. McGraw-Hill, New York, NY.

[11] Witten, I. H., Neal R. M., and Cleary J. G. 1987. Fast arithmetic coding using low-precision division. *Communications of the ACM* **30**, 520-540.