Brendan J. Frey and Geoffrey E. Hinton 1997. Efficient stochastic source coding and an application to a Bayesian network source model. *The Computer Journal* **40**, 157–165.

Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model

Brendan J. Frey
Geoffrey E. Hinton

Department of Computer Science
University of Toronto
6 King's College Road
Toronto, Canada, M5S 3H5

December 15, 1997

Abstract

In this paper, we introduce a new algorithm called "bits-back coding" that makes stochastic source codes efficient. For a given one-to-many source code, we show that this algorithm can actually be *more* efficient than the algorithm that always picks the shortest codeword. Optimal efficiency is achieved when codewords are chosen according to the Boltzmann distribution based on the codeword lengths. It turns out that a commonly used technique for determining parameters — maximum likelihood estimation — actually minimizes the bits-back coding cost when codewords are chosen according to the Boltzmann distribution. A tractable approximation to maximum likelihood estimation — the generalized expectation maximization algorithm — minimizes the bits-back coding cost. After presenting a binary Bayesian network model that assigns exponentially many codewords to each symbol, we show how a tractable approximation to the Boltzmann distribution can be used for bits-back coding. We illustrate the performance of bits-back coding using using nonsynthetic data with a binary Bayesian network source model that produces 2^{60} possible codewords for each input symbol. The rate for bits-back coding is nearly one half of that obtained by picking the shortest codeword for each symbol.

1 Introduction

Ordinary source codes map each input symbol, or block of symbols, to a unique codeword. This mapping is determined by an underlying source model whose parameters may be fixed by hand, estimated using a stored data set, or estimated on-line. In this paper we consider "one-to-many" source models that map each symbol to a probability distribution across multiple alternative codewords. At first sight, it appears that such models must be less efficient than one-to-one models. Whereas a one-to-one model can always pick the shortest codeword, the one-to-many model may sometimes pick longer codewords. The obvious way to eliminate this inefficiency is to define a second level code in which each codeword represents a disjunction of all the first-level codewords that stand for the same symbol. However, this is not possible when the sets of codewords that stand for the same symbol are exponentially large as often happens in the domains in which one-to-many source codes naturally arise. We describe a different way of completely eliminating the apparent inefficiency of one-to-many source models using stochastic coding. The advantage of this method is that it can be approximated quite well even when there are exponentially many codewords per symbol and can therefore eliminate most of the inefficiency caused by one-to-many source codes.

Our reason for being interested in one-to-many source models is that they arise naturally in many domains. We first describe a simple mixture of Gaussians example. This example is used to illustrate how our stochastic "bits-back" coding method can make a one-to-many source code optimally efficient, even though stochastic coding is not really required in this case because other, conceptually simpler methods are quite tractable. We then show that there is a tractable approximation to the optimal stochastic coding method, and that this approximation makes such source models efficient. Also, it leads to an objective function that can be used for fitting the parameters of such source models to observed data even though maximum likelihood estimation is computationally intractable. We present compression results using software-implemented bits-back coding and a simple toy problem. We then consider a much more complicated source model called a "binary Bayesian network", which naturally assigns exponentially many codewords to each input symbol. Parameterized Bayesian networks are capable of representing complex distributions and are therefore suitable for universal source coding. Using a simple image compression problem, we compare the performances of bits-back coding with a binary Bayesian network; shortest codeword selection with a binary Bayesian network; and UNIX `gzip`.

2 The mixture of Gaussians source model

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 simplicity.

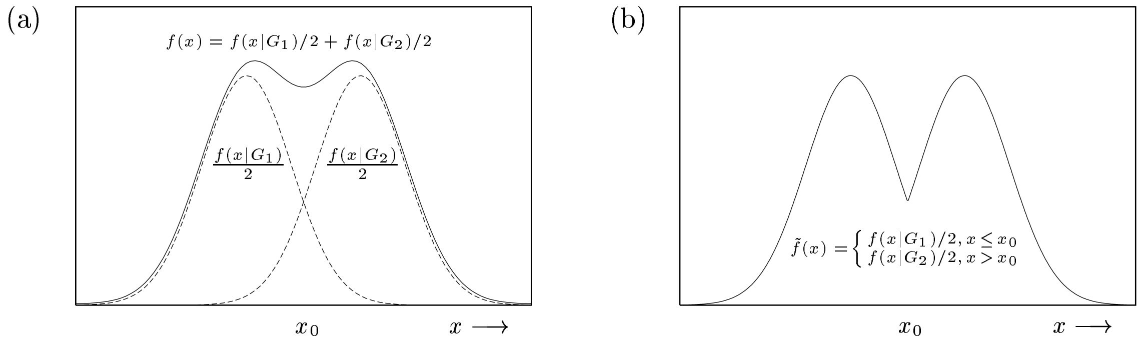

Figure 1: The most natural source model may produce multiple codewords for a given symbol. (a) shows a source with a single binary hidden variable which identifies from which Gaussian, G_1 or G_2, the symbol value x is sampled. Values of x near x_0 are likely to have come from either Gaussian. (b) shows 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.

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.

The obvious way around this problem is to use a one-to-one code that is based on the mixture distribution. That is, we assign a codeword to each symbol based on its total probability mass, obtained by summing the contributions from each Gaussian. We show that the same efficiency can be achieved by stochastically picking codewords from a one-to-many code. This may seem surprising, since for a given symbol both codewords in the one-to-many source code are longer than the codeword in the one-to-one source code. However, we will show that extra information can be communicated through the choice of codewords.

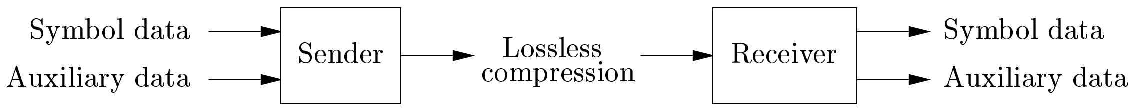

Figure 2: A scheme in which auxiliary data is communicated along with the primary symbol data in order to achieve optimal compression when the source code produces multiple codewords for a given symbol.

3 Getting bits back

The approach we take to solve this problem is motivated by the "bits-back" argument of Hinton and Zemel 1994 which they used to develop a Lyapunov function for machine learning. Since this paper was written, it has come to our attention that Wallace 1990 also developed this line of argument to construct minimum-length integer-length messages for use in minimum-message-length inference. By selecting codewords through the use of extra *auxiliary* information, this information can ride piggyback on the codewords for the primary data that we are encoding. 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 encoding is shown in figure 2. Notice that if the sender does not wish to communicate an extra stream of auxiliary data to the receiver, some of the primary symbol data can be set aside and used as auxiliary data. (It may need to be XORed with a pseudo-random bit stream to make it appear more random to the sender.)

Suppose in the Gaussian mixture example that a sender wishes to encode a truncated value, x', that is twice as likely under G_1 as it is under G_2, and that 2 bits are required to encode the truncated value under G_1. Including the single bit required to specify which Gaussian is being used, an optimal source code (possibly arithmetic) will thus have codewords with 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 whenever the sender must communicate the particular symbol x', the sender 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 $(l_1 + l_2)/2 = 3.5$ 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 choice of codeword 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 encoding x', 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. (It is important to note that this codeword specifies which of G_1 and G_2 is being used.) When decoding, the receiver reads off the bit that says which Gaussian was used and then determines the truncated value x' 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 is 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. We refer to this method of stochastic source coding as *bits-back coding*. It is important to note that the ratio of choices depends on the symbol being encoded. For example, if the truncated value is far to the right of x_0 in figure 1a, then picking the codewords equally often would be very inefficient, since the codeword under G_1 would be extremely long, making the benefit of the single recovered bit negligible. In this case the sender should pick G_1 much less often and, as a result the sender will read off only "part of a bit" from the auxiliary data queue to determine which codeword to use.

The efficiency of bits-back coding can be determined 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 $\ell(x, y)$ be the length of codeword y for a specific x, the average codeword length for x is

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

The average bits-back for x 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 $\mathcal{F}(x)$ that bits-back coding can achieve:

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

Since this quantity is analogous to the variational *Helmholtz free energy* from statistical physics (see textbook reference Thompson 1988), we refer to the effective average codeword length as the *free energy*.

Bits-back coding makes gains over shortest codeword selection by taking into account the existence of multiple codewords of similar length. It is easily proven from equations

2, 3 and 4 that the codeword selection distribution which minimizes the free energy is the Boltzmann distribution based on the codeword lengths:

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

We denote by "*" those quantities determined from the Boltzmann distribution. This distribution gives the optimal effective codeword length for the given one-to-many source code. The free energy for the Boltzmann codeword selection distribution is

$$\begin{aligned} F^*(x) = \mathcal{E}^*(x) - \mathcal{H}^*(x) &= \sum_y \frac{2^{-\ell(x,y)}}{\sum_{\hat{y}} 2^{-\ell(x,\hat{y})}} \ell(x,y) + \sum_y \frac{2^{-\ell(x,y)}}{\sum_{\hat{y}} 2^{-\ell(x,\hat{y})}} \log_2 \frac{2^{-\ell(x,y)}}{\sum_{\hat{y}} 2^{-\ell(x,\hat{y})}} \\ &= -\log_2 \sum_y 2^{-\ell(x,y)}. \end{aligned} \tag{6}$$

So, the optimal bits-back coding rate is equivalent to the rate obtained by mixing the probability masses associated with the codewords for input x.

In the above example, where for symbol x' we had $l_1 = 3$ bits and $l_2 = 4$ bits,

$$\begin{aligned} Q^*(G_1|x') &= 2^{-3}/(2^{-3} + 2^{-4}) = 2/3, \\ Q^*(G_2|x') &= 2^{-4}/(2^{-3} + 2^{-4}) = 1/3, \\ \mathcal{E}^*(x') &= \frac{2}{3}(3 \text{ bits}) + \frac{1}{3}(4 \text{ bits}) = 3.333 \text{ bits}, \\ \mathcal{H}^*(x') &= -\frac{2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3}) = 0.918 \text{ bits}, \\ \mathcal{F}^*(x') &= 3.333 \text{ bits} - 0.918 \text{ bits} = 2.415 \text{ bits}. \end{aligned} \tag{7}$$

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 $\hat{Q}(G_1|x') = \hat{Q}(G_2|x') = 0.5$.) If we directly mix the two codewords, we get a new codeword with length

$$-\log_2 \left[2^{-3} + 2^{-4}\right] = 2.415 \text{ bits}, \tag{8}$$

which is identical to the minimum free energy given above.

Suppose that instead of picking codewords according the Boltzmann distribution , the sender always chooses the shortest codeword. In this case, the number of extraneous bits relative to using the Boltzmann distribution is

$$\mathcal{F}^{\text{short}}(x) - \mathcal{F}^*(x) = \sum_y Q^{\text{short}}(y|x) \log_2 \frac{Q^{\text{short}}(y|x)}{Q^*(y|x)} = -\log_2 Q^*(y^{\text{short}}|x), \tag{9}$$

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

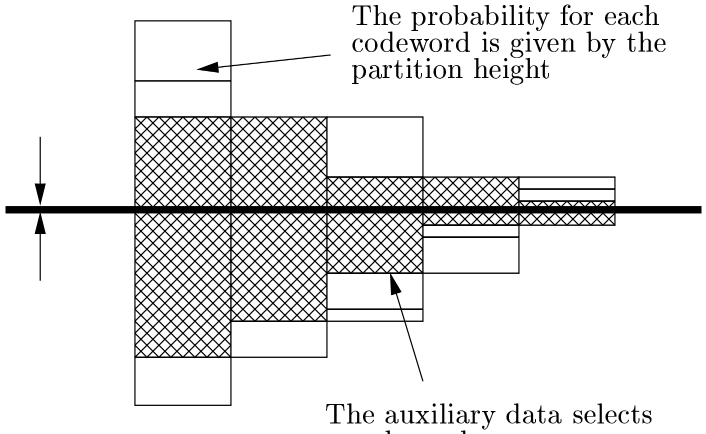

Figure 3: Feeding a random number into an arithmetic decoder with appropriate probabilities (shown by the partition heights within a column) selects codewords (shaded partitions), while at the same time conserving information.

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. As we shall see, the advantage of the bits-back approach is that it allows most of the inefficiency to be eliminated from one-to-many source codes even when the Boltzmann distribution cannot be computed exactly.

4 The bits-back coding algorithm

To implement the communication scheme shown in figure 2, we need a general method of recovering the auxiliary data bits from the codeword choices. In the example of a mixture of two Gaussians, we considered an input symbol for which selecting the two codewords equally often was nearly optimal, so that a single bit could be used for bits-back. If the codeword selection distribution is dyadic[1], it turns out that Huffman *decoding* (Huffman 1952) can be used to pick codewords. Here, we consider the case of an arbitrary codeword selection distribution.

In the general case of nondyadic codeword selection distributions, it is not so easy to see how random codeword choices can be made without losing auxiliary data information.

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

To address this problem, consider the operation of an arithmetic decoder (Rissanen and Langdon 1976; Witten *et. al.* 1987; Moffat *et. al.* 1995). 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 proportional 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).

By applying an arithmetic encoder to the sequence of decisions using the appropriate probabilities, we can regenerate the finite-precision number.

5 Relationship to maximum likelihood estimation and the expectation maximization algorithm

In general, a source model contains a set of parameters Θ that must be fixed by hand, estimated using a stored data set, or estimated on-line. Estimating these parameters is a difficult task when there are hidden variables. A popular statistical technique that is used to fit such models is *maximum likelihood estimation*. An algorithm known as the expectation maximization (EM) algorithm (Baum and Petrie 1966; Dempster *et. al.* 1977; Meng and Rubin 1992) is commonly used to obtain this estimate. Letting y represent the configuration of the hidden variables, maximum likelihood estimation assigns the following cost to each input symbol x:

$$\mathcal{C}(x) = -\log_2 P(x) = -\log \sum_y P(x, y). \tag{10}$$

If for each x and y, we can produce an optimal codeword with length $\ell(x, y) = -\log_2 P(x, y)$, then the maximum likelihood cost function can be expressed in terms of $\ell(x, y)$ as

$$\mathcal{C}(x) = -\log_2 \sum_y 2^{-\ell(x,y)} = \mathcal{F}^*(x), \tag{11}$$

where the last step follows from equation 6. So, the maximum likelihood cost function is equal to the optimal bits-back coding cost, indicating that maximum likelihood estimation minimizes the optimal bits-back coding cost. In contrast, maximum likelihood estimation *does not* minimize the communication cost associated with an encoder that always picks the shortest codeword.

Using Jensen's inequality, it is easily shown that the maximum likelihood cost function can be reformulated as

$$\mathcal{C}(x) = \sum_y Q^*(y|x) \log_2 \frac{Q^*(y|x)}{P(x,y)}, \quad \text{where} \quad Q^*(y|x) \equiv \frac{P(x,y)}{\sum_{\hat{y}} P(x,\hat{y})}. \tag{12}$$

The EM algorithm operates in an iterative fashion, alternately computing $Q^*(y|x)$ and then minimizing $\mathcal{C}(x)$ with respect to the parameters of $P(x,y)$. Often, maximum likelihood estimation is not tractable because the computation of $Q^*(y|x)$ is intractable. In these cases, it is possible to use an approximation to maximum likelihood estimation which can be thought of as a generalization of the EM algorithm (Neal and Hinton 1993). This method approximates $Q^*(y|x)$ with a more tractable distribution $\hat{Q}(y|x)$, which is continuously updated so as to be close (in the sense of relative entropy) to $Q^*(y|x)$. If this approximate distribution is used as the codeword selection distribution, then it turns out that the generalized EM algorithm attempts to minimize the bits-back coding cost. As with exact maximum likelihood estimation, this method *does not* attempt to minimize the communication cost associated with an encoder that always picks the shortest codeword.

6 Experimental results for a toy problem

In this section, we present a simple parameterized source model. Then we describe how an artificial source is produced by randomly selecting model parameters and then simulating data. The model is simple enough that the hidden variables *can* be summed over so that it is possible to compute the entropy of the artificial source. Then, model parameters are determined by applying maximum likelihood estimation to the simulated data set. After discussing the theoretical free energy, we present results when this model is used in software-implemented versions of bits-back 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 bits-back coding. In fact, HMMs are usually simple enough that the hidden variables (state) can tractably be integrated out. As a result, bits-back coding is not normally worthwhile for HMMs; however, the HMM provides a good toy problem, since the performance of bits-back coding can be compared with the true entropy of the artificial source. 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; see Rabiner 1989 for a tutorial exposition. It is sufficient to note that the model parameters can be estimated from a data set using a maximum likelihood estimation technique called the Baum-Welch algorithm (Baum and Petrie 1966) and also that for a given symbol sequence the model can be used to produce a codeword that represents both the state sequence $\mathbf{y}$ of the HMM and the symbol sequence $\mathbf{x}$.

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 compression 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 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 Rabiner 1989 we find that the entropy of this artificial 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 some 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 20.6 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 11.8 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 a software implementation of a coding algorithm that always chooses the shortest codeword with a software implementation of a bits-back 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. To orient the reader, we also include the performance of the UNIX `gzip` utility which was executed with the `-best` option. The file sizes and rates (communicated bits/string, less recovered auxiliary data bits) are given in table 1. The rate for the bits-back coding algorithm comes close to the optimum value given by the source entropy. The rate for shortest codeword selection is nearly twice that of the bits-back coding rate. An implementational note: the bits-back coding software yields rates that are indistinguishable from the theoretical free energies for the corresponding models, to at least one decimal place.

[2]To find the shortest codeword for a given string, we use the Viterbi algorithm (Viterbi and Omura 1979).

Table 1: Rate comparisons for software-implemented source codes on the HMM data.

	Rate (bits/string)
Original ascii file	88.0
Shortest codeword selection using the estimated HMM	20.6
Bits-back coding using the estimated HMM	11.8
`gzip -best`	16.0
Bits-back coding using the artificial source	9.3
Source entropy	9.3

7 An application to binary Bayesian networks

Although the above example illustrates how bits-back coding works, equivalent performance can be obtained for the HMM by directly mixing together the codewords for a given input sequence. In the remainder of this paper, we focus on a model for which the multiple codewords corresponding to an input cannot be directly mixed. The sigmoidal Bayesian network described in this section is a highly flexible adaptive source model that is capable of representing complex distributions over binary data vectors. For this reason, it is well-suited to universal source coding. This type of model was shown by Frey *et. al.* 1996 to produce competitive density estimators for several synthetic and nonsynthetic data sources, when fit using the algorithm described in the following section.

In order to model binary data vectors, we will assume that each vector is produced by a set of unobserved, or "hidden", binary-valued causes. For each binary data vector $\mathbf{x}$ and for each binary configuration $\mathbf{y}$ of the causes, the model can produce a codeword. We require that the code be instantaneous; *ie.*, that the codewords satisfy the Kraft-inequality:

$$\sum_{\mathbf{x},\mathbf{y}} 2^{-\ell(\mathbf{x},\mathbf{y})} \leq 1, \tag{13}$$

where $\ell(\mathbf{x},\mathbf{y})$ is the length of the codeword for data vector $\mathbf{x}$ and configuration $\mathbf{y}$. In fact, we do not restrict these codewords to be of integer length. The model is used to produce these codewords of noninteger length by providing a probability $P(\mathbf{x},\mathbf{y})$ for each $\mathbf{x}-\mathbf{y}$ pair. These probabilities are used in conjunction with arithmetic coding to produce "codewords" of length

$$\ell(\mathbf{x},\mathbf{y}) = -\log_2 P(\mathbf{x},\mathbf{y}). \tag{14}$$

In order be suitable for arithmetic coding, the model must be able to provide not only the probability mass $P(\mathbf{x},\mathbf{y})$, but a placement of this mass with respect to the probability masses of all the other possible data vectors and configurations. So, if the binary data vector $\mathbf{x}$ contains $n_{\mathbf{x}}$ bits and the configuration vector $\mathbf{y}$ contains $n_{\mathbf{y}}$ bits, then there are

$2^{n_\mathbf{x}+n_\mathbf{y}}$ possible probability masses that must be properly arranged before codewords can be assigned. A Bayesian network simultaneously provides a structured probability model as well as the means to break apart this set of probability masses into smaller pieces.

For notational simplicity, let $\mathbf{s}$ be the concatenation of the configuration vector and the data vector:

$$\mathbf{s} = (\mathbf{y}, \mathbf{x}), \quad ie. \quad (s_1, s_2, \ldots, s_{n_\mathbf{x}+n_\mathbf{y}}) = (y_1, y_2, \ldots, y_{n_\mathbf{y}}, x_1, x_2, \ldots, x_{n_\mathbf{x}}), \tag{15}$$

where x_i are the binary elements of $\mathbf{x}$, y_i are the binary elements of $\mathbf{y}$, and s_i are the binary elements of $\mathbf{s}$. A Bayesian network is a model that decomposes the joint distribution $P(\mathbf{s}) = P(\mathbf{x}, \mathbf{y})$ in a way that is congruous with Bayes rule:

$$\begin{aligned} P(\mathbf{s}) &= P(s_1)P(s_2|s_1)P(s_3|s_1, s_2)\ldots P(s_{n_\mathbf{x}+n_\mathbf{y}}|s_1, \ldots, s_{n_\mathbf{x}+n_\mathbf{y}-1}), \\ P(\mathbf{s}) &= \prod_{i=1}^{n_\mathbf{x}+n_\mathbf{y}} P(s_i|s_j, \forall j < i). \end{aligned} \tag{16}$$

Any distribution can be written in this form, but the important quality of a Bayesian network is that each component $P(s_i|s_j, \forall j < i)$ is computed with relative ease. In our application, we compute them using a set of parameters Θ in the following way:

$$P(s_i|s_j, \forall j < i) = \begin{cases} 1/(1 + \exp[-\theta_i - \sum_{j<i} \theta_{ij} s_j]) & \text{if } s_i = 1, \\ 1 - 1/(1 + \exp[-\theta_i - \sum_{j<i} \theta_{ij} s_j]) & \text{if } s_i = 0. \end{cases} \tag{17}$$

This form essentially relates the probability that s_i has the value 1 *linearly* to the values of $s_j, \forall j < i$. The logistic function, $1/(1 + exp[\,])$, is simply used to produce a probability between 0 and 1. We usually constrain some of the θ_{ij}'s to be zero. These constraints can be represented graphically if we assign one node to each element of $\mathbf{s}$ and draw a directed edge from node s_j to s_i only if $j < i$ *and* $\theta_{ij} \neq 0$. An example of a top-down layered network is shown in figure 4a. This network uses two layers of binary causes and one layer of binary values at the bottom for the data vector. The middle layer is directly connected to the bottom layer, so it is easy to see how those binary causes can influence the data vector. On the other hand, the top layer is *not* directly connected to the bottom layer, so it may seem that the top layer is redundant. However, the top layer of causes may be needed to capture any covariance between the middle-layer causes, since without the top layer we have $P(s_4, s_5, s_6) = P(s_4)P(s_5)P(s_6)$.

Arithmetic coding can now quite easily be used to encode an $\mathbf{x}$ - $\mathbf{y}$ pair to produce a codeword with length $\ell(\mathbf{x}, \mathbf{y})$. The sender simply traverses the elements $s_1, s_2, \ldots, s_{n_\mathbf{x}+n_\mathbf{y}}$, encoding each binary value using its probability computed from equation 17.

Given a data vector $\mathbf{x}$, the sender would ideally pick a configuration $\mathbf{y}$ using some auxiliary data bits and the Boltzmann distribution given in equation 5. From equation 14, the Boltzmann distribution for the binary Bayesian network is given by

$$Q^*(\mathbf{y}|\mathbf{x}) \equiv \frac{P(\mathbf{x}, \mathbf{y})}{\sum_{\hat{\mathbf{y}}} P(\mathbf{x}, \hat{\mathbf{y}})}, \tag{18}$$

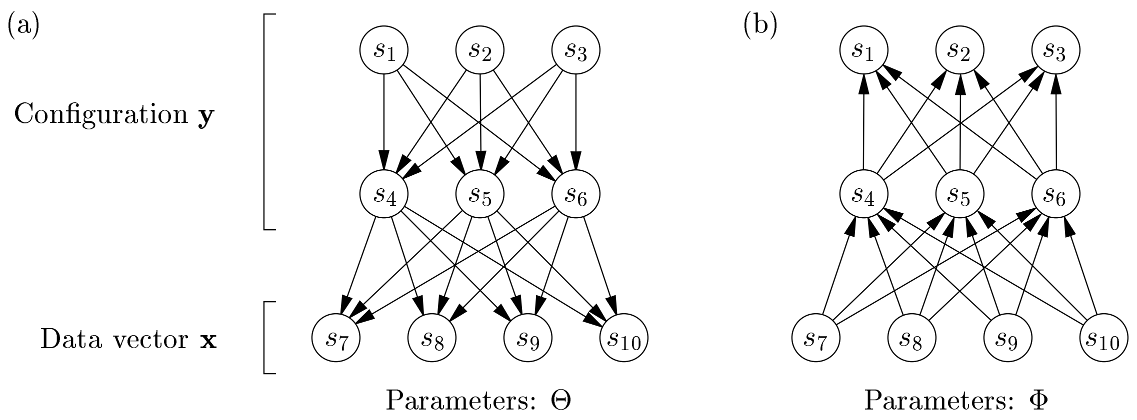

Figure 4: (a) A Bayesian network can be used to model a binary data vector $\mathbf{x}$ using a vector of hidden binary causes $\mathbf{y}$. However, for a given $\mathbf{x}$ the Boltzmann distribution over $\mathbf{y}$ cannon be tractably computed. (b) shows a bottom-up Bayesian network that produces a distribution $Q(\mathbf{y}|\mathbf{x})$ that can approximate the Boltzmann distribution.

and the free energy is given by

$$\mathcal{F}^*(\mathbf{x}) = \sum_{\mathbf{y}} Q^*(\mathbf{y}|\mathbf{x}) \log_2 \frac{Q^*(\mathbf{y}|\mathbf{x})}{P(\mathbf{x},\mathbf{y})}. \tag{19}$$

For the models that we are interested in, $n_{\mathbf{y}}$ is large — at least twenty — so the sum in the denominator of equation 18 will have at least 2^{20} terms. Obviously, this sum is intractable so the sender cannot use the exact Boltzmann distribution to pick codewords. Shortest codeword selection is also not an easy task in this case. An exhaustive search would again require the consideration of 2^{20} possible codewords, and there is no guarantee for how well a more greedy algorithm would perform.

8 An approximation to the Boltzmann distribution for the binary Bayesian network

Instead of computing the Boltzmann distribution exactly, we use a probability density model $Q(\mathbf{y}|\mathbf{x})$ to approximate $Q^*(\mathbf{y}|\mathbf{x})$. In fact, this model is of the same form as the binary Bayesian network that defines $P(\mathbf{x}, \mathbf{y})$, except that the connectivity is reversed:

$$Q(s_i|s_j, \forall j > i) = \begin{cases} 1/(1 + \exp[-\phi_i - \sum_{j>i} \phi_{ij} s_j]) & \text{if } s_i = 1, \\ 1 - 1/(1 + \exp[-\phi_i - \sum_{j>i} \phi_{ij} s_j]) & \text{if } s_i = 0, \end{cases} \tag{20}$$

where Φ is the set of parameters for this bottom-up network. Note that in this bottom-up network the probability for s_i depends on variables with indices *greater* than i, whereas in the top-down network it depends on variables with indices *less* than i. Again, some of these parameters may be constrained to be zero as shown graphically in figure 4b. This sort of bottom-up model forms the basis of a new type of "neural network", as described by Hinton *et. al.* 1995 and Dayan *et. al.* 1995. We will show in the next section how this model can be used to pick codewords using auxiliary data. The free energy given by this approximation can be compared to the minimum free energy given by the Boltzmann distribution:

$$\mathcal{F}(\mathbf{x}) - \mathcal{F}^*(\mathbf{x}) = \sum_{\mathbf{y}} Q(\mathbf{y}|\mathbf{x}) \log_2 \frac{Q(\mathbf{y}|\mathbf{x})}{Q^*(\mathbf{y}|\mathbf{x})}. \tag{21}$$

This is the Kullback-Leibler pseudo-distance (aka. relative entropy) between the codeword selection distribution and the Boltzmann distribution. It is always non-negative and yields the number of extraneous bits in the compression as a result of not using the Boltzmann distribution.

9 Bits-back coding using a binary Bayesian network

In this section, we describe how bits-back coding can be applied to a binary Bayesian network source model. Then, we present compression results when the model is fit to a set of images using an approximation to the generalized EM algorithm described above. We compare the compression efficiency of the one-to-many bits-back source coding algorithm with the one-to-one source code obtained using approximate shortest codeword selection, and also with the UNIX `gzip` utility. The set of results describe in this section is more realistic than the set of results described for the HMM for two reasons. First, the number of bits in the configuration vector $\mathbf{y}$ for this experiment is $n_{\mathbf{y}} = 60$, and there is no way to efficiently mix all 2^{60} configurations. Second, we use a nonsynthetic data set that consists of 100 000 normalized and quantized 8×8 binary images of handwritten digits made available by the US Postal Service Office of Advanced Technology (see figure 5).

We now show how bits-back coding can be used to encode the data vector (s_7, s_8, s_9, s_{10}) using the top-down and bottom-up networks shown in figure 4. The sender first computes $Q(s_6|s_7, s_8, s_9, s_{10})$ using equation 20. An arithmetic *decoder* uses the auxiliary data bits as input in conjunction with this probability to determine the value of s_6. Once this value is determined, the sender computes $Q(s_5|s_6, s_7, s_8, s_9, s_{10})$ (which in this case does not depend on s_6) and uses the arithmetic decoder to determine s_5. Note that in general the configuration must be determined bit by bit. For example, in the current case it is impossible for the sender to compute $Q(s_3|s_4, s_5, s_6, s_7, s_8, s_9, s_{10})$ without having determined a value for s_4. This bit by bit procedure continues until the configuration $(s_1, s_2, s_3, s_4, s_5, s_6)$ has been completely determined. Then, the sender uses an arithmetic *encoder* to produces a "codeword" that represents both the configuration and the data vector. This is done in a similar bit by bit fashion but in the top-down direction, using equation 17 to compute the bit probabilities.

Given the codeword, the receiver uses an arithmetic *decoder* to determine, in a bit by bit fashion, first the configuration bits $(s_1, s_2, s_3, s_4, s_5, s_6)$ and then the data vector bits (s_7, s_8, s_9, s_{10}). Next, the receiver recovers the auxiliary data bits that the sender used to choose a codeword. The receiver uses the bottom-up network and equation 20 to compute $Q(s_6|s_7, s_8, s_9, s_{10})$ — note that since the data vector has been losslessly communicated, this probability will be the same one that the sender used to encode s_6. An arithmetic *encoder* uses the already decoded value of s_6 as input in conjunction with this probability in order to recover the auxiliary data that was used to choose the value of s_6. The receiver continues to recover auxiliary data bits, using the probabilities given by equation 20 in conjunction with the already decoded configuration bits, until the bits-back for s_1, s_2, s_3, s_4, s_5 and s_6 have been obtained.

This procedure is repeated for each new data vector that is to be communicated. With useful auxiliary data, this scheme empowers the sender to communicate bits-back, obtaining bits-back coding. With a good codeword selection distribution, we can achieve near-optimal effective compression for the given source code.

The binary Bayesian network that we use as a source model has three hidden layers of binary causes and one bottom-layer data vector that consists of $n_{\mathbf{x}} = 64$ binary elements. From top to bottom, the three layers of causes have 16 elements, 20 elements, and 24 elements, giving a total of $n_{\mathbf{y}} = 60$ binary causes. Both the top-down and the bottom-up networks are fully-connected from layer to layer, but have no connections within each layer (*cf.*, figure 4a and 4b). The model is fit to a training set that constists of 100 000 images, using an approximation to the generalized EM algorithm that is introduced in Hinton *et. al.* 1995 and reviewed more extensively in Frey *et. al.* 1996. This estimation method makes use of the bottom-up network to approximate the Boltzmann distribution that would ideally be used to perform maximum likelihood estimation based on the cost function in equation 12.

For the purpose of comparing bits-back coding with a one-to-one source code, we approximate shortest codeword selection, since an exact implementation would require a full search of all 2^{60} codewords for each data vector. Instead, for each data vector we picked the shortest in a list of ten stochastically chosen codewords. By producing these codewords using the bottom-up network with pseudo-random numbers instead of auxiliary data, we biased the list to contain short codewords. (If the codewords had been chosen completely at random, they would most often be ridiculously long.)

A comparison of the rates obtained using approximate shortest codeword selection and

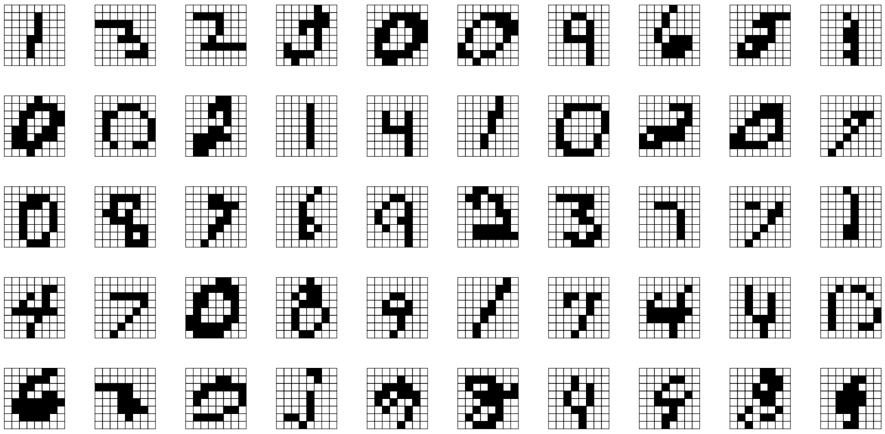

Figure 5: Examples of 8x8 binary images of handwritten digits.

Table 2: Rate comparisons for software-implemented source codes on the binary digit data.

	Rate (bits/image)
Original binary file	64
Shortest codeword selection using the estimated model	60
Bits-back coding using the estimated model	33
`gzip -best`	39

bits-back coding with the estimated binary Bayesian network, as well as the rate obtained by the UNIX `gzip` utility with the `-best` option, are given in table 2. (Although the UNIX `gzip` utility is not really meant for image compression, we include it as a reference point for the reader.) The rate for shortest codeword selection is again significantly higher than the rate for bits-back coding, indicating that a significant practical savings can be made by using the proposed approach.

10 Conclusions

We have introduced the concept of efficient stochastic source codes that use bits-back. Further, it is apparent that the standard statistical technique of maximum likelihood estimation as well as a large body of popular methods for fitting models with hidden variables — the expectation maximization variants — produce source models for which shortest codeword selection is suboptimal in compression efficiency whereas bits-back coding is optimal or nearly optimal. We have shown that a practical bits-back coding algorithm can be implemented and that it does indeed outperform a shortest codeword selection algorithm in compression performance.

It may seem that one drawback of bits-back coding is the need for a stream of auxiliary data. Often, for example when compressing a computer file, there is no separate set of data that can be conveniently used as auxiliary data. However, this problem can be avoided simply by setting aside some of the primary source data and using it as auxiliary data. (It may need to be XORed with a pseudo-random bit stream to make it appear more random to the sender.) While this does eliminate the need for extra auxiliary data, it is not optimal because the source data is uncoded. We are currently exploring more optimal ways of eliminating the need for extra auxiliary data.

Simple bits-back coding software is available at `http://www.cs.toronto.edu/~frey`, or by ftp at `ftp://ftp.cs.utoronto.ca/pub/frey/bbc.tar.Z`.

Acknowledgements

We thank Radford Neal for helpful discussions and two anonymous referees for useful comments. Our bits-back coding software uses arithmetic coding software provided by Radford Neal. We are grateful to the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre for financial support.

References

Baum, L. E., and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics* **37**, 1559–1563.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz machine. *Neural Computation* **7**, 889–904.

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.

Frey, B. J., Dayan, P., and Hinton, G. E. (1997). A simple algorithm that discovers efficient perceptual codes. In M. Jenkin and L. R. Harris (eds), *Computational and biological mechanisms of visual coding*, Cambridge University Press, Cambridge.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science* **268**, 1158–1161.

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 (eds), *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, San Francisco.

Huffman, D. A. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers* **40**, 1098–1101.

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 (eds), *Bayesian Statistics 4*. Clarendon Press, Oxford.

Moffat, A., Neal, R. M., and Witten I. H. (1995). Arithmetic coding revisited. In J. A. Storer and M. Cohn (eds), *Proceedings of the Data Compression Conference 1995*, IEEE Computer Society Press, Los Alamitos.

Neal, R. M., and Hinton, G. E. (1993). A new view of the EM algorithm that justifies incremental and other variants. Unpublished manuscript. Available over the internet by ftp at `ftp://ftp.cs.utoronto.ca/pub/radford/em.ps.Z`.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* **77**, 257–286.

Rissanen, J., and Langdon, G. G. (1976). Arithmetic coding. *IBM Journal of Research and Development* **23**, 149–162.

Thompson, C. J. (1988). *Classical Equilibrium Statistical Mechanics.* Clarendon Press, Oxford.

Viterbi, A. J., and Omura, J. K. (1979). *Principles of Digital Communication and Coding.* McGraw-Hill, New York.

Wallace, C. S. (1992). Classification by minimum-message-length inference. In S. G. Akl *et. al.* (eds), *Lecture Notes in Computer Science* Vol. 468. Springer, Berlin.

Witten, I. H., Neal R. M., and Cleary J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM* **30**, 520–540.