Lossless compression based on the Sequence Memoizer

Jan Gasthaus[†] Frank Wood[‡] Yee Whye Teh[†]

† Gatsby Computational Neuroscience Unit, UCL, London, UK
‡ Department of Statistics, Columbia University, New York, USA

Abstract

In this work we describe a sequence compression method based on combining a Bayesian nonparametric sequence model with entropy encoding. The model, a hierarchy of Pitman-Yor processes of unbounded depth previously proposed by Wood et al. [2009] in the context of language modelling, allows modelling of long-range dependencies by allowing conditioning contexts of unbounded length. We show that incremental approximate inference can be performed in this model, thereby allowing it to be used in a text compression setting. The resulting compressor reliably outperforms several PPM variants on many types of data, but is particularly effective in compressing data that exhibits power law properties.

1 Introduction

We consider compression of sequences using a predictive model that incrementally estimates a distribution over what symbol comes next from the preceding sequence of symbols. As our predictive model we use the sequence memoizer (SM) [Wood et al., 2009], a nonparametric Bayesian model for sequences of unbounded complexity. This model is combined with an entropy coder, for instance the arithmetic coder of Witten et al. [1987], to yield a method for compressing sequence data. The main contribution of this paper is to develop an efficient approximate incremental inference algorithm for the SM that renders it suitable for use as a compressor's predictive model.

At the algorithmic level, the proposed method is somewhat similar to the unbounded context length variants of the well known PPM and CTW algorithms [Cleary and Teahan, 1997; Willems, 1998]. At a conceptual level, however, our approach is quite different: We take a Bayesian approach, treating the distributions over next symbols as latent variables on which we place a hierarchical nonparametric prior, and predicting the next symbol by averaging over the posterior distribution (conditioned on the sequence observed thus far). The prediction problem thus essentially becomes an incremental inference problem, where the posterior distribution is incrementally updated after observing each symbol.

We first present our algorithm in Section 2. The model, its implied prior assumptions, and the derivation of the algorithm as an approximate inference scheme in the model will

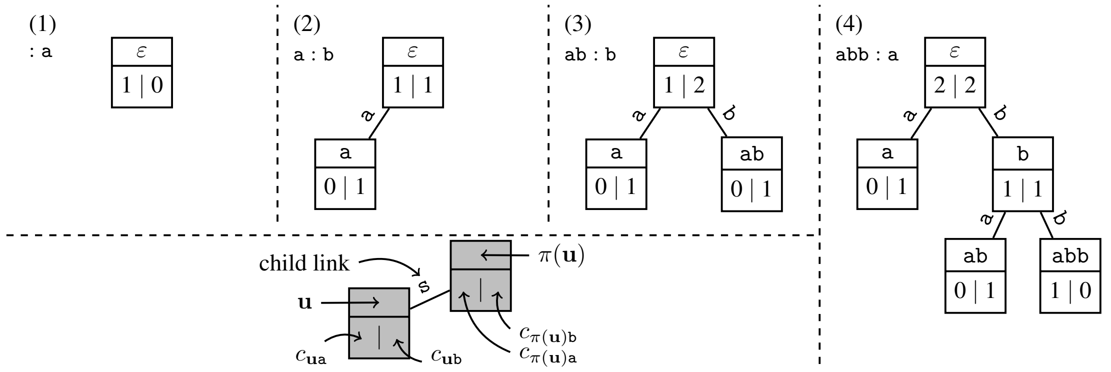

Figure 1: Illustration of the tree construction algorithm for the input string abba. Each step shows the state of the tree after inserting the current context and updating the counts for the observed symbol. Shown below the context at each node are the counts $c_{\mathbf{ua}}$ (left) and $c_{\mathbf{ub}}$ (right) as produced by the algorithms.

be discussed in Sections 3 and 4. In Section 5 we present state-of-the-art text compression performances on the Calgary corpus.

2 Algorithm

In this section we will give a complete description of the algorithm (Algorithm 1) for computing the predictive probability for the next symbol given the sequence of symbols observed (or decoded) so far and for updating the model after observing the next symbol. Let us first define the notation that is used throughout the paper. Let Σ denote the set of symbols (alphabet) over which strings are defined. We will denote strings by bold letters $\mathbf{u}, \mathbf{x} \in \Sigma^\star$, where the input sequence is always denoted by $\mathbf{x}$. The length of a string $\mathbf{u}$ is denoted by $|\mathbf{u}|$, and the empty string (of length 0) by ε. Further, let $\sigma(\mathbf{u})$ denote the longest proper suffix of $\mathbf{u}$ (i.e. $\mathbf{u}$ with the first symbol removed).

Similar to CTW and PPM, our algorithm maintains information about the already observed input sequence $\mathbf{x}_{1:i}$ in the form of a context tree, where each node corresponds to some context (i.e. a substring) in $\mathbf{x}_{1:i}$ and is associated with a set of adjustable parameters. Having processed the input up to position i, the algorithm proceeds as follows in order to predict (and subsequently encode) the next symbol $\mathbf{x}_{i+1}$: 1) the context tree structure is updated to include the context $\mathbf{x}_{1:i}$; 2) the predictive probabilities $\mathrm{P}(s|\mathbf{x}_{1:i})$ are computed for all $s \in \Sigma$, and arithmetic coding used to encode $\mathbf{x}_{i+1}$; 3) the parameters of the tree are updated to account for the new observation $\mathbf{x}_{i+1}$. The function PLUMP/DEPLUMP in Algorithm 1 performs all of these steps (PLUMP stands for Power Law Unbounded Markov Prediction; DEPLUMP is the compressor for obvious reasons).

The context tree is essentially a (compressed) suffix tree of the reversed string $\mathbf{x}_{i:-1:1}$, and can straightforwardly be constructed sequentially as shown in Figure 1. The necessary update is given in the function INSERTCONTEXTANDRETURNPATH.

Algorithm 1 PLUMP/DEPLUMP

```
1: procedure PLUMP/DEPLUMP($\mathbf{x}$)
2:     Initialize context tree $\mathcal{T}_\varepsilon$ and counts $\mathcal{S}_\varepsilon$
3:     for $i = 0, \dots, |\mathbf{x}| - 1$ do
4:         $\mathcal{T}_{\mathbf{x}_{1:i+1}}, (\mathbf{u}_0, \dots, \mathbf{u}_P) \leftarrow$ INSERTCONTEXTANDRETURNPATH$(\mathcal{T}_{\mathbf{x}_{1:i}}, \mathbf{x}_{1:i})$
5:         $(p_{-1}, \dots, p_P) \leftarrow$ PATHPROBABILITY$(\mathcal{S}_{\mathbf{x}_{1:i}}, (\mathbf{u}_0, \dots, \mathbf{u}_P))$
6:         Use probabilities $p_P$ to encode/decode $\mathbf{x}_{i+1}$
7:         $\mathcal{S}_{\mathbf{x}_{1:i+1}} \leftarrow$ UPDATEPATH $(\mathcal{S}_{\mathbf{x}_{1:i}}, (\mathbf{u}_0, \dots, \mathbf{u}_P), (p_{-1}, \dots, p_P), \mathbf{x}_{i+1})$
8:     end for
9: end procedure
10: function INSERTCONTEXTANDRETURNPATH($\mathcal{T}_{\mathbf{x}_{1:i}}, \mathbf{x}_{1:i}$)
11:     Traverse tree to find the node $\mathbf{x}_{j:k}$ that shares the longest suffix with $\mathbf{x}_{1:i}$.
12:     if $\mathbf{x}_{j:k}$ is suffix of $\mathbf{x}_{1:i+1}$ then
13:         Insert $\mathbf{x}_{1:i}$ into the tree as a child of $\mathbf{x}_{j:k}$
14:     else
15:         Let $\mathbf{x}_{l:k}$ be the longest suffix of both $\mathbf{x}_{1:i}$ and $\mathbf{x}_{j:k}$.
16:         Insert $\mathbf{x}_{l:k}$ into the tree in the position of $\mathbf{x}_{j:k}$
17:         Make $\mathbf{x}_{j:k}$ and $\mathbf{x}_{1:i+1}$ children of $\mathbf{x}_{j:k}$
18:     end if
19:     return updated tree $\mathcal{T}_{\mathbf{x}_{1:i+1}}$ and traversed path $(\mathbf{u}_0, \dots, \mathbf{u}_P)$ from root to $\mathbf{x}_{1:i}$
20: end function
21: function PATHPROBABILITY($\mathcal{S}_{\mathbf{x}_{1:i}}, (\mathbf{u}_0, \dots, \mathbf{u}_P), s$)
22:     $p_{-1} \leftarrow H(s)$
23:     for $j = 0, \dots, P$ do
24:         for $s \in \Sigma$ do
25:             $p_j(s) \leftarrow$ evaluate equation (1) with $\mathbf{u} = \mathbf{u}_j$ and $\mathrm{P}(s|\pi(\mathbf{u}), \mathcal{S}_{\mathbf{x}_{1:i}}) = p_{j-1}(s)$
26:         end for
27:     end for
28:     return $(p_{-1}, p_0, \dots, p_P)$
29: end function
30: function UPDATEPATH($\mathcal{S}_{\mathbf{x}_{1:i}}, (\mathbf{u}_0, \dots, \mathbf{u}_P), (p_{-1}, \dots, p_P), s = \mathbf{x}_{i+1}$)
31:     for $j = P, \dots, -1$ do  ▷ Update counts for each context $\mathbf{u}_j$ on the path, bottom up
32:         $c_{\mathbf{u}_j s} \leftarrow c_{\mathbf{u}_j s} + 1$
33:         if $t_{\mathbf{u}_j s} = 0$ then $t_{\mathbf{u}_j s} = 1$
34:         if use probabilistic updates then
35:             Increment $t_{\mathbf{u}_j s}$ with probability $\frac{d_{\mathbf{u}_j} t_{\mathbf{u}_j \cdot} p_{j-1}(s)}{c_{\mathbf{u}_j s} - d_{\mathbf{u}_j} t_{\mathbf{u}_j s} + d_{\mathbf{u}_j} t_{\mathbf{u}_j \cdot} p_{j-1}(s)}$
36:         end if
37:         if $t_{\mathbf{u}_j s}$ unchanged then break
38:     end for
39:     return updated state $\mathcal{S}_{\mathbf{x}_{1:i+1}}$
40: end function
```

For each context $\mathbf{u}$ in the context tree, the algorithm maintains two sets of counts, $\{c_{\mathbf{u}s}\}_{s\in\Sigma}$ and $\{t_{\mathbf{u}s}\}_{s\in\Sigma}$, as well as a scalar parameter $d_{\mathbf{u}}$. These counts are initialized to 0 for all contexts and symbols s. Given these sets of counts for all contexts $\mathbf{u} \in \Pi_{\mathbf{x}_{1:i}}$ (which we will in the following jointly refer to as the *state* of the model $\mathcal{S}_{\mathbf{x}_{1:i}}$), the predictive probability for a symbol s following context $\mathbf{u}$ is given by

$$\mathrm{P}(s|\mathbf{u}, \mathcal{S}_{\mathbf{x}_{1:i}}) = \begin{cases} \frac{c_{\mathbf{u}s} - d_{\mathbf{u}} t_{\mathbf{u}s}}{c_{\mathbf{u}\cdot}} + \frac{d_{\mathbf{u}} t_{\mathbf{u}\cdot}}{c_{\mathbf{u}\cdot}} \mathrm{P}(s|\pi(\mathbf{u}), \mathcal{S}_{\mathbf{x}_{1:i}}) & \text{if } c_{\mathbf{u}\cdot} \neq 0; \\ \mathrm{P}(s|\pi(\mathbf{u}), \mathcal{S}_{\mathbf{x}_{1:i}}) & \text{otherwise,} \end{cases} \tag{1}$$

where $c_{\mathbf{u}\cdot} = \sum_{s\in\Sigma} c_{\mathbf{u}s}$, $t_{\mathbf{u}\cdot} = \sum_{s\in\Sigma} t_{\mathbf{u}s}$, and $\pi(\mathbf{u})$ is the parent of $\mathbf{u}$ in the context tree. At the root of the tree we have $\mathrm{P}(s|\pi(\varepsilon), \mathcal{S}_{\mathbf{x}_{1:i}}) = 1/|\Sigma|$.

Given a path $(\mathbf{u}_0, \ldots, \mathbf{u}_P)$, the function PATHPROBABILITY computes the probability of each symbol s in all contexts along the path. While for prediction we are just interested in the predictive probability vector p_P for the full context $\mathbf{x}_{1:i}$ at the end of the path, the other probabilities are needed as input for the parameter update function UPDATEPATH, which updates the counts in $\mathcal{S}_{\mathbf{x}_{1:i}}$ for all nodes along $(\mathbf{u}_0, \ldots, \mathbf{u}_P)$ in light of the observation $\mathbf{x}_{i+1}$. This function can either make deterministic or probabilistic updates. The motivation behind both types of updates will be described in section 4. Intuitively, the updates can be understood in the following way: Every time a new observation is added to the model counts are incremented for some (random) distance up the tree. This sharing of counts is the way observations in related contexts reinforce each other.

The discount parameters $d_{\mathbf{u}}$ are not independent for each context, but rather are functions of underlying parameters $d_{|\mathbf{u}|}$ that depend on the length of the context—this relationship will be made clear in Section 4. The underlying parameters can either be fixed a priori or can be initialized to some value and subsequently be updated alongside the counts, e.g. by taking small steps along the gradient of (1) with respect to $d_{|\mathbf{u}|}$ after observing $\mathbf{x}_{i+1}$.

The computational complexity of the described method is $\mathcal{O}(N^2)$ where N is the length of the input sequence. For each input symbol we have to perform constant-time computations along the path from the root to the context. For the i-th symbol, this path is of length i in the worst case, leading to an overall quadratic complexity. For typical text inputs however, the paths are much shorter. The space complexity is linear in the length of the input sequence: Each new observation adds at most two nodes to the tree, and nodes require constant space.

3 Probabilistic Model

Our compressor is derived from an underlying probabilistic model which we call the sequence memoizer. This is a hierarchical Bayesian nonparametric model composed of Pitman-Yor processes originally conceived of as a model for languages (sequences of words). In this section we briefly describe the model and refer the interested reader to [Teh, 2006; Wood et al., 2009].

The model describes the conditional probability of each symbol s following each context $\mathbf{u}$ using a latent variable $G_{\mathbf{u}}(s)$. Collecting the variables into a vector, $G_{\mathbf{u}} = [G_{\mathbf{u}}(s)]_{s\in\Sigma}$ is a probability vector (non-negative entries summing to one), and the full (infinite) set of latent variables in the model is $\mathcal{G} = \{G_{\mathbf{u}}\}_{\mathbf{u}\in\Sigma^*}$. The joint probability of $\mathbf{x}$ and $\mathcal{G}$ is simply:

$$P(\mathbf{x}, \mathcal{G}) = P(\mathcal{G}) \prod_{i=0}^{|\mathbf{x}|-1} G_{\mathbf{x}_{1:i}}(\mathbf{x}_{i+1}) \tag{2}$$

where the rightmost term is the probability of each symbol conditioned on the sequence thus far, and $P(\mathcal{G})$ is the prior over the variables. We will describe the prior in the rest of this section. Taking a Bayesian approach and marginalizing out $\mathcal{G}$, we get a distribution $P(\mathbf{x})$ with which we can compress $\mathbf{x}$. Section 4 describes how the algorithm in Section 2 is an approximation of this ideal Bayesian approach.

3.1 Pitman-Yor Process

The Pitman-Yor process (PYP) [Pitman and Yor, 1997], denoted $\mathcal{PY}(d, H)$, is a distribution over probability vectors.[1] It is parameterized by a *discount parameter* $d \in (0, 1)$ and a probability vector H called the *base vector*. If $G \sim \mathcal{PY}(d, H)$ is a Pitman-Yor distributed random probability vector, then the base vector is simply its mean $\mathrm{E}[G(s)] = H(s)$ while the discount parameter is related to its variance $\mathrm{Var}[G(s)] = (1 - d)H(s)(1 - H(s))$, for each $s \in \Sigma$. Intuitively, G is "similar" to H, and the parameter d control how G varies around H. Another interesting aspect of the PYP is that it exhibits power-law properties, making them very successful in modelling data with such properties (such as natural languages) [Teh, 2006].

A better understanding of the PYP can be obtained by way of the *Chinese restaurant process (CRP)* [Pitman, 2002]. Initialize two sets of counts $\{c_s, t_s\}_{s\in\Sigma}$ to 0, and consider the following generative process: Draw $y_1 \sim H$ and set $c_{y_1} = t_{y_1} = 1$; for $n = 2, 3, \ldots$ and each $s \in \Sigma$, with probability $\frac{c_s - dt_s}{c_\cdot}$ set $y_n = s$ and increment c_s, and with probability $\frac{dt_\cdot}{c_\cdot} H(s)$ set $y_n = s$ and increment *both* c_s and t_s. The process generates a random sequence of symbols $y_1, y_2, \ldots \in \Sigma$. These symbols are dependent through the updates on the counts (e.g. if $y_1 = s$ we will more likely see $y_2 = s$ as well). Coming back to the PYP, since G is a probability vector, we can treat it as a multinomial distribution over Σ. Consider a sequence of iid draws $y_1, y_2, \ldots \sim G$. The randomness in G induces dependencies among the y_n's once G is marginalized out, and the resulting distribution over the sequence $y_1, y_2, \ldots$ is precisely captured by the Chinese restaurant process.

3.2 Sequence Memoizer

The sequence memoizer (SM) [Wood et al., 2009] is distribution over the infinite set $\mathcal{G} = \{G_{\mathbf{u}}\}_{\mathbf{u}\in\Sigma^*}$ of probability vectors reposed on a hierarchical Bayesian model consisting

[1]In this paper we describe a simplified Pitman-Yor process. Firstly, Pitman-Yor processes are distributions over *distributions* over an arbitrary probability space (here our space is Σ, a finite alphabet set). Secondly, Pitman-Yor processes typically include a third parameter called the concentration or strength parameter. In this three parameter guise the Pitman-Yor process is a generalization of the Dirichlet distribution, the more commonly encountered distribution over probability vectors, which is obtained when $d = 0$ and the concentration parameter is positive. Here the concentration parameter is set to 0 instead, this is sometimes referred to as the *normalized stable process*.

of PYPs. Succinctly, we can describe the SM as follows:

$$G_\varepsilon \mid d_0, H \sim \mathcal{PY}(d_0, H) \tag{3a}$$
$$G_\mathbf{u} \mid d_{|\mathbf{u}|}, G_{\sigma(\mathbf{u})} \sim \mathcal{PY}(d_{|\mathbf{u}|}, G_{\sigma(\mathbf{u})}) \qquad \forall \mathbf{u} \in \Sigma^+ \tag{3b}$$

where H is a base vector (in the following assumed to be uniform) and $d_{|\mathbf{u}|}$ are the discount parameters. The structure of the model is an unbounded-depth tree, with each node indexed by a context $\mathbf{u} \in \Sigma^*$, labelled by the probability vector $G_\mathbf{u}$, and with parent given by $\sigma(\mathbf{u})$.

The structure encodes the prior knowledge that the predictive distributions over the subsequent symbols in different contexts are similar to each other, with contexts sharing longer suffixes being more similar to each other. In other words, the later symbols in a context are more important in predicting the subsequent symbol. Also, by virtue of using PYPs, the SM encodes an assumption that sequences of symbols have power-law properties.

To make computations in the SM tractable, it is essential to reduce the size of the model. Firstly, we can marginalize out all $G_\mathbf{u}$ not associated with the data in $\mathbf{x}$. This reduces the size of the tree from infinite to quadratic in $|\mathbf{x}|$. Secondly, we can marginalize out all non-branching nodes in the resulting tree, which further reduces the size to linear in $|\mathbf{x}|$. The resulting tree is the context tree as described in Section 2. The second reduction can be performed analytically due to a theorem of Pitman [1999], which in our situation simply states: if $G_1|G_0 \sim \mathcal{PY}(d_1, G_0)$ and $G_2|G_1 \sim \mathcal{PY}(d_2, G_1)$ then marginally $G_2|G_0 \sim \mathcal{PY}(d_1 d_2, G_0)$ is a Pitman-Yor process as well with modified discount parameters. Further details of this reduction can be found in [Wood et al., 2009]. Note that the algorithm in [Wood et al., 2009] requires the entire observation sequence to be known to construct the context tree efficiently. However, this is not necessary as the context tree can straightforwardly be constructed incrementally as described in Section 2. This is essential for text compression since the decoder does not have access to the entire sequence until it has finished decoding.

4 Prediction, Inference and Estimation

Exact Bayesian computations in the SM model, such as finding the predictive distribution $\mathrm{P}(s|\mathbf{x}_{1:i})$, are intractable. In Section 2 we described an incremental algorithm for estimating $\mathrm{P}(s|\mathbf{x}_{1:i})$ based on viewing the Pitman-Yor distributed random vectors $G_\mathbf{u}$ in terms of the CRP as described in Section 3.1. In this view, the random vectors $G_\mathbf{u}$ are not explicitly represented; instead, the marginal distribution of draws from $G_\mathbf{u}$ is captured by the counts $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}_{s\in\Sigma}$ of the CRP. Because the $G_\mathbf{u}$'s are hierarchically coupled in the SM (3), the CRPs are coupled as well, leading to a joint generative process for the counts $\mathcal{S}_{\mathbf{x}_{1:i}} = \{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}_{\mathbf{u}\in\Pi_{\mathbf{x}_{1:i}}, s\in\Sigma}$ in all contexts on the tree. Given $\mathcal{S}_{\mathbf{x}_{1:i}}$, the predictive probability of the symbol s following $\mathbf{x}_{1:i}$ is given by (1), which is simply a recursive application of the predictive probabilities in the CRPs associated with the contexts on the path from $\mathbf{x}_{1:i}$ to the root. The quantity we are actually interested in, $\mathrm{P}(s|\mathbf{x}_{1:i})$, can then be approximated by averaging (1) over posterior samples of the CRP counts $\mathcal{S}_{\mathbf{x}_{1:i}}$ conditioned on $\mathbf{x}_{1:i}$.

The task thus becomes obtaining samples from the posterior over the CRP counts conditioned on the observations $\mathbf{x}_{1:i}$. In our previous work [Teh, 2006; Wood et al., 2009] these samples were obtained using Markov chain Monte Carlo methods, which is too computationally intensive for the sequential setting considered here. In this paper we take a different approach: at each iteration i we simply update the counts associated with the new observation of $\mathbf{x}_{i+1}$ in the CRP associated with context $\mathbf{x}_{1:i}$, and do not resample all the other counts. In the sequential Monte Carlo literature, this is simply a particle filter with only one particle, and corresponds to the UPDATEPATH function when probabilistic updates are used; we call this variant 1PF.

To obtain a better approximation to the posterior, the particle filter can be run with multiple particles. Surprisingly, this results in a rather negligible improvement (see Section 5). There are two reasons for this. Firstly, as [Teh, 2006] noted, the posterior is unimodal, so it is easy for the particle filter to obtain a sample close to the mode and this single sample is likely to perform well. Secondly, much of the predictive ability of the SM comes from the hierarchical sharing of counts which is already present in a single sample.

The alternative non-probabilistic count update described in Algorithm 1 is an additional approximation: Instead of maintaining the counts $t_{\mathbf{u}s}$, we constrain them to be 1 whenever $c_{\mathbf{u}s} > 0$ and 0 otherwise. As noted by [Teh, 2006], this approximation yields predictive probabilities equal to the ones that would be obtained using interpolated Kneser-Ney, a state-of-the-art smoothing algorithm for language models; we call this variant UKN (for unbounded-depth Kneser-Ney). In our experiments in Section 5 we find that in the text compression setting this approximation also works very well.

5 Experiments

In order to evaluate DEPLUMP in terms of compression performance on various types of input sequences we use it to make incremental next symbol predictions on the Calgary corpus – a well known compression benchmark corpus consisting of 14 files of different types and varying lengths. The measure used for comparing the different algorithms is the *average log-loss* $\ell(\mathbf{x}_{1:N}) = -\frac{1}{N}\sum_{i=1}^{N}\log_2 p(x_i|\mathbf{x}_{1:i-1})$ which corresponds to the average number of bits per symbol required to encode the sequence using an optimal code. As entropy coding can achieve this limit up to a small additive constant, it is virtually equivalent to the average number of bits per symbol required by the compressed file. For all our experiments we treat the input files as sequences of bytes, i.e. with a 256 alphabet size.[2]

The results are shown in Table 1. For comparison, we also show the results of two PPM variants and one CTW variant in the final three columns. PPM* was the first PPM variant to use unbounded-length context, and the results for PPM-Z are (to our knowledge) among the best published results for a PPM variant on the Calgary corpus.

There are several observations that can be made here: first, the compression results for UKN, and 1PF are comparable, with no consistent advantage for any single approach

[2]In all experiments the per-level discount parameters were initialized to the values $d_{0:10} = (0.05, 0.7, 0.8, 0.82, 0.84, 0.88, 0.91, 0.92, 0.93, 0.94, 0.95)$ and $d_\infty = 0.95$ (chosen empirically). They were then optimized by gradient ascent in the predictive probability, interleaved with the count updates. Additionally, in order to overcome problems with the model becoming overconfident after long runs of the same symbol, predictions were made using a mixture model, where the predictions from the leaf node in the tree are mixed with predictions from the root (with a weight of 10^{-2} for the root). An alternative solution to the problem would be to pre-process the input using run-length encoding.

		DEPLUMP		PPM		CTW
File	Size	1PF	UKN	PPM*	PPMZ	CTW
bib	111261	1.73	**1.72**	1.91	1.74	1.83
book1	768771	**2.17**	2.20	2.40	2.21	2.18
book2	610856	**1.83**	1.84	2.02	1.87	1.89
geo	102400	**4.40**	4.40	4.83	4.64	4.53
news	377109	**2.20**	2.20	2.42	2.24	2.35
obj1	21504	**3.64**	3.65	4.00	3.66	3.72
obj2	246814	2.21	**2.19**	2.43	2.23	2.40
paper1	53161	2.21	**2.20**	2.37	2.22	2.29
paper2	82199	**2.18**	2.18	2.36	2.21	2.23
pic	513216	0.77	0.82	0.85	**0.76**	0.80
progc	39611	2.23	**2.21**	2.40	2.25	2.33
progl	71646	1.44	**1.43**	1.67	1.46	1.65
progp	49379	1.44	**1.42**	1.62	1.47	1.68
trans	93695	1.21	**1.20**	1.45	1.23	1.44
avg.		**2.12**	2.12	2.34	2.16	2.24
w. avg.		**1.89**	1.91	2.09	1.93	1.99

Table 1: Compression performance in terms of average log-loss (average bits per character under optimal entropy encoding) for the Calgary corpus. Boldface type indicates best performance. Ties are resolved in favour of lowest computational complexity. The results for PPM* (PPM with unbounded-length contexts) are copied from [Cleary and Teahan, 1997] and are actual compression rates, while the results for PPMZ are average log-losses obtained using a modified version of PPMZ 9.1 under Linux [Peltola and Tarhio, 2002] (which differ slightly from the published compression rates). The results for CTW were taken from [Willems, 2009].

on all files. While 1PF has a slight advantage over UKN on the larger text files `book1` and `book2`, this advantage is mostly lost on other file types. This means that the most computationally efficient approach (1PF) can be chosen without having to suffer a trade-off in terms compression performance.

In addition to these experiments, we performed experiments with several variants of the basic algorithm: with and without context mixing, with and without gradient updates to the discounts, and with a model variant where the discount parameters are independent (i.e. not shared) for each context. We also tested the algorithm with more than one particle, in which case the predictions were averaged over particles. Two main observations could be made from these experiments: 1) context mixing is essential if files contain long runs (such as `pic`), and 2) neither using independent discount parameters nor using more particles improves performance consistently. Using 100 particles yields ≈ 0.02 bps improvement on the large text files (`book`, and `paper`), but no improvement on the other files.

In addition to the experiments on the Calgary corpus, compression performance was also evaluated on two other benchmark corpora: the Canterbury corpus [Arnold and Bell, 1997] and the 100 MB excerpt of an XML text dump of the English version of Wikipedia used in the Large Text Compression Benchmark [Mahoney, 2009] and the Hutter Prize compression challange [Hutter, 2006]. On the Canterbury corpus, the results were consistently better then the best reported results, with the exception of two binary files.

On the Wikipedia excerpt, the UKN algorithm (without context mixing) achieved a log-loss of 1.66 bits/symbol amounting to a compressed file size of 20.80 MB. While this is significantly worse than 16.23 MB achieved by the currently best PAQ-based compressors (but on par with non-PAQ approaches), it demonstrates that the described approach can scale to sequences of this length.

Finally, we explored the performance of the algorithm when using a larger alphabet. In particular, we used an alphabet of 16-bit characters to compress Big-5 encoded Chinese text. On a representative text file, the Chinese Union version the the bible, we achieved a log-loss of 4.35 bits per Chinese character, which is significantly better than the results reported by Wu and Teahan [2007] (5.44 bits).

6 Discussion

We presented a new compression algorithm based on the predictive probabilities of a hierarchical Bayesian nonparametric model called the sequence memoizer (SM). We showed that the resulting compressor, DEPLUMP, compresses a variety of signals as well or better than PPM*, PPMZ and CTW. The reasons for this include both the fact that DEPLUMP uses unbounded context lengths and the fact that DEPLUMP employs an underlying probabilitistic model, SM, that explicitly encodes power law assumptions. SM enables DEPLUMP to use the information available in the unbounded length contexts effectively, whereas for PPM* the extension to unbounded depth did not yield consistent improvement [Bunton, 1997].

While we show that DEPLUMP surpasses the compression performance of PPM's best variants, it should should be noted that PPM has also recently been surpassed by the context-mixing PAQ family of compressors [Mahoney, 2005] and PAQ compression performance currently exceeds (in general) that of DEPLUMP as well. Context-mixing is a term used by the PAQ community; we would suggest that the phrase predictive model mixing more accurately describes what context-mixing compressors do. This means, roughly, that PPM can be (and seems already to be—the literature is somewhat obtuse on this point) embedded into PAQ as one of the models that are mixed together. Correspondingly, improvements to PPM compression should translate into improvements to PAQ compression, albeit perhaps more modestly since PAQ relies upon other predictive models as well. In general, PAQ utilizes a diverse set of models, each of which uses a different definition of context and generalization relationships between contexts. Integrating such ideas into the nonparametric Bayesian framework presented here remains an exciting opportunity for future research.

The use of a predictive model for compression has a long tradition, with PPM, CTW and PAQ being stellar examples. Latent variables that capture regularities in the data, as applied in this paper, have previously been used in the compression setting [Hinton and Zemel, 1994], but have seen less application due to the computational demand of approximating often intractable posterior distributions. Fortunately, in this paper we were able to derive efficient and effective approximate inference algorithms yielding state-of-the-art compression results.

We conclude this paper with an interesting thought concerning hierarchical PYP models like the SM. These models were originally intended for modeling sequences of words coming from a large (even infinite) vocabulary [Teh, 2006]. In this paper we have used basically the same model under very different circumstances: the symbol set is small (256 instead of $\gg 10000$), but typical repeated context lengths are much longer (10-15 instead of 3-5). It is surprising that the same model can handle both extremes well and points to a sense that it is a natural model for discrete sequential data. It also bodes well for our current work applying DEPLUMP to large alphabet texts, e.g. of Chinese and Japanese.

References

Arnold, R. and Bell, T. (1997). A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference, 1997. Proceedings*, pages 201–210.

Bunton, S. (1997). Semantically Motivated Improvements for PPM Variants. *The Computer Journal*, 40(2/3).

Cleary, J. G. and Teahan, W. J. (1997). Unbounded length contexts for PPM. *The Computer Journal*, 40:67–75.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and Helmholtz free energy. *Advances in Neural Information Processing Systems 6*, pages 3–10.

Hutter, M. (2006). Prize for compression human knowledge. URL: `http://prize.hutter1.net/`.

Mahoney, M. (2009). Large text compression benchmark. URL: `http://www.mattmahoney.net/text/text.html`.

Mahoney, M. V. (2005). Adaptive weighing of context models for lossless data compression. Technical report, Florida Tech. Technical Report CS-2005-16, 2005.

Peltola, H. and Tarhio, J. (2002). URL: `http://cs.hut.fi/u/tarhio/ppmz`.

Pitman, J. (1999). Coalescents with multiple collisions. *Annals of Probability*, 27:1870–1902.

Pitman, J. (2002). Combinatorial stochastic processes. Technical Report 621, Department of Statistics, University of California at Berkeley.

Pitman, J. and Yor, M. (1997). The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *Annals of Probability*, 25:855–900.

Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the Association for Computational Linguistics*, pages 985–992.

Willems, F. M. J. (1998). The context-tree weighting method: Extensions. *IEEE Transactions on Information Theory*, 44(2):792–798.

Willems, F. M. J. (2009). CTW website. URL: `http://www.ele.tue.nl/ctw/`.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540.

Wood, F., Archambeau, C., Gasthaus, J., James, L., and Teh, Y. W. (2009). A stochastic memoizer for sequence data. In *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada.

Wu, P. and Teahan, W. J. (2007). A new PPM variant for Chinese text compression. *Natural Language Engineering*, 14(3):417–430.