Improving PPM with dynamic parameter updates

Christian Steinruecken, Zoubin Ghahramani, David MacKay
Department of Engineering, University of Cambridge

Abstract

This article makes several improvements to the classic PPM algorithm, resulting in a new algorithm with superior compression effectiveness on human text. The key differences of our algorithm to classic PPM are that (A) rather than the original escape mechanism, we use a generalised blending method with explicit hyper-parameters that control the way symbol counts are combined to form predictions; (B) different hyper-parameters are used for classes of different contexts; and (C) these hyper-parameters are updated dynamically using gradient information.

The resulting algorithm (PPM-DP) compresses human text better than all currently published variants of PPM, CTW, DMC, LZ, CSE and BWT, with runtime only slightly slower than classic PPM.

1 Introduction

The classic PPM algorithm by Cleary and Witten (1984) compresses sequences of symbols one symbol at a time, by gradually learning context-dependent conditional probability distributions. At the heart of every PPM-like algorithm is a data structure that stores symbol occurrence counts for each context, in a way that allows these counts to be accessed efficiently to compute the conditional symbol distributions. The primary difference between different variants of PPM is the way these counts are updated and combined to form predictions: these two details jointly define the *probabilistic model* that determines the algorithm's compression effectiveness.[1]

This paper gives an explicit form of the probabilistic model of the classic PPM algorithm, and then proposes several modifications, each of which increase the compression effectiveness on human text. The runtime costs of this new method are comparable to those of other PPM variants, and none of our proposed changes increases the memory requirements of the algorithm.

The resulting finite-depth algorithm (PPM-DP) compresses human text better than any published PPM-variant, including Charles Bloom's PPMZ (Bloom 1998) and Dmitry Shkarin's PPMII (Shkarin 2001a, 2002). It also beats CTW (Willems et al. 1993, 1995; Willems 1998), BWT (Burrows and Wheeler 1994; Seward 2010), DMC (Cormack 1993), CSE (Dubé and Beaudoin 2010), and LZMA (Pavlov 2003, 2011) on human text.

[1] Of course all compression algorithms define a probability distribution over the input objects they compress, at least implicitly. Suppose that an algorithm C maps input sequences S to compressed output sequences $C(S)$. Then C's implicit probability distribution P_C over input sequences S is given by $P_C(S) = \frac{1}{Z} \cdot 2^{-|C(S)|}$, where $|C(S)|$ is the length of the compressed output sequence, and Z is a constant that normalises the probabilities to unity.

2 Background

PPM-like algorithms use arithmetic coding to compress and decompress sequences one symbol at a time, using predictive probability distributions $P(x_n \mid x_1 \ldots x_{n-1})$ that depend on the preceding symbols. Each time a symbol is encoded or decoded, the algorithm updates its internal model to improve the symbol predictions for the remainder of the sequence.

A characteristic feature of PPM algorithms is that these predictive symbol distributions are computed hierarchically from context-dependent symbol occurrence counts $\mathcal{M}$, which are collected from the input sequence during compression / decompression. These counts are collected separately for different contexts, where the context of a symbol (at a given position n) is the sequence of symbols immediately preceding it (or a finite suffix thereof). For example, the length-3 context of symbol x_n is the sequence $(x_{n-3}, x_{n-2}, x_{n-1})$. All PPM variants use a designated data structure (such as a trie) to be able to access these counts quickly. Finite-depth variants of PPM consider contexts up to some maximum length $D_{\max}$.

Let $\mathcal{X}$ denote the alphabet of input symbols (possibly including a special `EOF` symbol for marking the end of the sequence). For any given context $\mathbf{s}$, let $\mathcal{M}_{\mathbf{s}}(x)$ denote how often a given symbol x was counted, and let $|\mathcal{M}_{\mathbf{s}}|$ denote the total number of symbols counted. It is important to note that the quantity $\mathcal{M}_{\mathbf{s}}(x)$ is *not* generally equal to the number of times the subsequence $(\mathbf{s} :: x)$ occurred in the sequence observed so far: the way symbols are counted and how their counts are combined to form predictions vary among PPM algorithms, and implicitly define the algorithm's probabilistic model.[2]

This paper proposes several changes to the original PPM algorithm, which are summarised as follows:

- Replacing the escape mechanism with *blending* of predictions from all available context depths. Blending was investigated in detail by Bunton (1997). We describe a generalised version of blending in section 4, in which a small number of hyper-parameters control the blending of predictions from different-depth contexts.

- Choosing separate hyper-parameters for different classes of contexts, grouped together by the number of unique symbols observed in the context, and by context-depth. This technique is described in section 5.

- Adjusting the hyper-parameters of each context class during compression, using the analytically computed gradients of the log probability of the data. Such a technique was used in the Deplump compressor by Bartlett and Wood (2011). This technique is described in section 6.

The conjunction of these techniques define a new algorithm named PPM-DP, whose compression effectiveness is demonstrated in section 7.

We begin by giving an explicit (and generalised) form of the probabilistic model of Cleary and Witten's PPM, including symbol exclusions and update exclusions by Moffat (1990).

[2] The most common rule for counting symbols is the "update exclusions" rule of Moffat (1990), which states that symbols are always counted in the longest matching context, but in shorter contexts are only counted once for every unique longer context. This rule is used in this paper.

3 The probabilistic model of PPM's escape mechanism

The original PPM algorithm by Cleary and Witten (1984) formed predictive symbol distributions using a method called the *escape mechanism.* The idea behind this mechanism is to predict the next symbol using only the count in the longest matching context if the symbol was observed at least once, and to back off to a shorter context otherwise.

The probabilistic model induced by this escape mechanism is stated below. We generalise the original escape mechanism slightly to include two hyper-parameters: a strength parameter α, and a discount parameter β. Some classic PPM variants correspond to particular settings of these hyper-parameters. For example, $\alpha = 1$ and $\beta = 0$ recovers Cleary and Witten's PPMA, while setting $\alpha = 0$ and $\beta = \frac{1}{2}$ produces Howard's PPMD (Howard 1993). The permitted range of β is the interval $[0, 1]$, and α must take a value between $-\beta$ and $+\infty$.

PPM's predictive distribution over the next symbol (assuming that all counts $\mathcal{M}_{\mathbf{s}}$ are up to date) can then be recursively expressed as follows:

$$P_{\mathbf{s}\backslash\mathcal{R}}(x) \;:=\; \begin{cases} \dfrac{\mathcal{M}_{\mathbf{s}}(x) - \beta}{T_{\mathbf{s}}^{\backslash\mathcal{R}} + \alpha} & \text{if } x \in \mathcal{M}_{\mathbf{s}} \text{ and } x \notin \mathcal{R} \\[2ex] \dfrac{U_{\mathbf{s}}^{\backslash\mathcal{R}} \cdot \beta + \alpha}{T_{\mathbf{s}}^{\backslash\mathcal{R}} + \alpha} \cdot P_{\mathrm{suf}(\mathbf{s})\backslash\mathcal{M}_{\mathbf{s}}}(x) & \text{otherwise.} \end{cases} \tag{1}$$

where the recursion starts with $\mathbf{s}$ being the longest supported context, and $\mathcal{R}$, the set of excluded symbols, is initially empty. The function suf($\mathbf{s}$) denotes the longest proper suffix of $\mathbf{s}$. For example, suf(`TEA`) = `EA`, and suf(`T`) = ε, where ε is the empty sequence. $T_{\mathbf{s}}^{\backslash\mathcal{R}}$ is the total number of symbol occurrences in $\mathcal{M}_{\mathbf{s}}$ (including repetitions) after excluding all symbols in $\mathcal{R}$:

$$T_{\mathbf{s}}^{\backslash\mathcal{R}} \;:=\; \sum_{x \in \mathcal{M}_{\mathbf{s}}} \mathcal{M}_{\mathbf{s}}(x) \cdot \mathbb{1}[x \notin \mathcal{R}] \tag{2}$$

and $U_{\mathbf{s}}^{\backslash\mathcal{R}}$ is the number of unique symbols in $\mathcal{M}_{\mathbf{s}}$ that do not occur in $\mathcal{R}$:

$$U_{\mathbf{s}}^{\backslash\mathcal{R}} \;:=\; \sum_{x \in \mathcal{X}} \mathbb{1}[x \in \mathcal{M}_{\mathbf{s}}] \cdot \mathbb{1}[x \notin \mathcal{R}]. \tag{3}$$

P_{ε} is the top-level (unigram) distribution, and $P_{\mathrm{suf}(\varepsilon)}$ is defined to be uniform (or some other suitable base distribution over the input alphabet $\mathcal{X}$). $P_{\mathbf{s}\backslash\mathcal{R}}$ denotes the adaptive symbol distribution for context $\mathbf{s}$ that excludes all symbols in set $\mathcal{R}$. (Note that the adaptive distributions $P_{\mathbf{s}}$ and summary multisets $\mathcal{M}_{\mathbf{s}}$ are changing objects whose predictions for symbol x_{N+1} depend on the preceding symbols $x_1 \ldots x_N$; this conditional dependence is left implicit for ease of notation.)

The escape mechanism appeals because of its simplicity and computational convenience, but its probabilistic model is somewhat difficult to reason about. Also, its compression effectiveness on human text is inferior to the *blending* method described in section 4; see e.g. the work by Bunton (1997) for comparisons of these two methods.

4 Blending

The classic PPM algorithm can be modified to use other forms of probability estimation, different from the original "escape mechanism" and its derivatives. The approach taken in this paper combines the observations from all context depths, *blending* their probability distributions (rather than switching between them and applying exclusion rules). In particular, we use the following construction:

$$P_{\mathbf{s}}(x) \;:=\; \frac{\mathcal{M}_{\mathbf{s}}(x) - \beta}{|\mathcal{M}_{\mathbf{s}}| + \alpha} \cdot \mathbb{1}[x \in \mathcal{M}_{\mathbf{s}}] \;+\; \frac{U_{\mathbf{s}}\beta + \alpha}{|\mathcal{M}_{\mathbf{s}}| + \alpha} \cdot \begin{cases} \dfrac{1}{|\mathcal{X}|} & \text{if } \mathbf{s} = \varepsilon \\[2ex] P_{\mathrm{suf}(\mathbf{s})}(x) & \text{otherwise,} \end{cases} \tag{4}$$

where $\mathbb{1}[x \in \mathcal{M}_{\mathbf{s}}]$ equals 1 if symbol x was observed at least once in context $\mathbf{s}$ (and 0 otherwise). $\mathcal{M}_{\mathbf{s}}$ is the multiset of symbol counts for context $\mathbf{s}$, and $U_{\mathbf{s}}$ denotes the number of unique symbols in $\mathcal{M}_{\mathbf{s}}$:

$$U_{\mathbf{s}} \;=\; \sum_{x \in \mathcal{X}} \mathbb{1}[x \in \mathcal{M}_{\mathbf{s}}]. \tag{5}$$

Equation (4) defines a probabilistic model that corresponds to a generalised form of "interpolated Kneser–Ney smoothing" (Chen and Goodman 1998) with a uniform base distribution. This model can also be considered an approximate sequential construction of a Pitman–Yor process (Teh 2006).

A blending PPM can be implemented straightforwardly by replacing PPM's escape mechanism with equation (4). There are no `ESC` symbols when blending is used: symbol probabilities are always calculated by visiting contexts of all depths.

Computing the cumulative symbol distributions of a blending PPM (as required for arithmetic coding) is computationally slightly more expensive compared to a PPM that uses the escape mechanism. However, the computational overhead is not prohibitive, and there may be several approaches for speeding up this computation, e.g. careful caching, or using approximations to blending like PPMII's information inheritance mechanism (Shkarin 2001a).

The predictive symbol distributions of a generalised blending PPM, as defined in equation (4), depend on two hyper-parameters α and β. It is worth pointing out that the settings of these hyper-parameters strongly affect the compression effectiveness of the model. How these hyper-parameters should be set can be determined empirically, or using an optimisation algorithm; for example, for English text, setting $\alpha = \frac{1}{2}$ and $\beta = \frac{3}{4}$ seems to work reasonably well (Steinruecken 2014).

We remark that although equation (4) uses the same two hyper-parameters as in equation (1), they affect compression effectiveness slightly differently in the two models; settings of α and β that yield good compression with the escape mechanism will not work well with blending, and vice versa.

5 Distinguishing different classes of context

Most PPM algorithms use the same parameter values for all contexts, i.e. one (often hard-wired) pair of hyper-parameters that govern the predictions for all contexts. One side-effect of this practice is, for example, the well-known problem that increasing the maximum context depth $D_{\max}$ can worsen (rather than improve) compression effectiveness. This same problem limits the effectiveness of unbounded depth PPM variants such as PPM*. Luckily, it turns out that this problem is easy to fix.

Instead of setting these hyper-parameters to fixed values that are shared by all contexts, we recommend setting the hyper-parameters differently for different *classes* of context. In particular, we recommend grouping contexts into classes based on their *depth* in the context tree (i.e.

the context length $|\mathbf{s}|$), and based on their *fanout* (the number of unique symbols $U_{\mathbf{s}}$ observed in the context). Each such group of contexts then shares one pair of hyper-parameters $(\alpha_{fd}, \beta_{fd})$, where d is the context's depth and f the context's fanout.[3]

The predictive distributions (4) remain the same, except that each instance of α and β is now looked up in a matrix of $F \times D$ entries. Only D different depths and F different fanouts are distinguished: for all contexts whose depth d exceeds D, the parameter value of the largest distinguished depth is used. The same rule applies for contexts whose fanout is larger than F. We write $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ for the two $(F \times D)$-sized matrices of parameter values. For good results on human text, in our experience, F should be at least 5, and D should be between 7 and D_{max}.

A key remaining question is of course how all these hyper-parameters should be set. If it was difficult to find good settings for a *single pair* of hyper-parameters, then setting $F \times D$ pairs of hyper-parameters might be even more challenging. In section 6, we address this problem using analytically derived gradients of the symbol distribution (with respect to the hyper-parameters).

Historical notes. The idea of using depth-dependent discount parameters may have originated with Chen and Goodman's versions of Kneser–Ney smoothing, in particular the "interpolated Kneser–Ney" method from section 4.1.6 of their technical report (Chen and Goodman 1998). Depth-dependent discount parameters are also used in the unbounded-depth PPM-variant named "Deplump" (Gasthaus et al. 2010; Bartlett and Wood 2011), and in the Sequence Memoizer of Wood et al. (2009) on which the Deplump compressor is based. Fanout-dependent parameters appear in section 4.1.7 of Chen and Goodman's report (Chen and Goodman 1998), and appear to be used (at least implicitly) in PPMII by Shkarin (2001a).

6 Setting the hyper-parameters

For any given input sequence $x_1 \dots x_N$, the optimal settings of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are those that minimise $\log_2 1/P(x_1 \dots x_N \mid \boldsymbol{\alpha}, \boldsymbol{\beta})$, the compressed output length. To find these settings, it is helpful to obtain analytic gradients of the output length with respect to the hyper-parameters.

The gradients of the output length can be computed additively from the gradients of the conditional symbol distributions:

$$\nabla \log_2 \frac{1}{P(x_1 \dots x_N)} = \sum_{n=1}^{N} \nabla \log_2 \frac{1}{P(x_n \mid x_1 \dots x_{n-1})} \tag{6}$$

where $P(x_n \mid x_1 \dots x_{n-1})$ was defined in (4). These gradients can be computed efficiently, with not much more effort than required for computing the probability mass.

6.1 Gradients

Writing $P_{\mathbf{s}}$ as an abbreviation for $P_{\mathbf{s}}(x_{N+1} \mid x_1 \dots x_N, \boldsymbol{\alpha}, \boldsymbol{\beta})$, and switching to natural logarithms, the gradients of the conditional symbol probabilities with respect to a discount parameter β_{fd} are:

$$\frac{\partial \log P_{\mathbf{s}}}{\partial \beta_{fd}} = \frac{1}{P_{\mathbf{s}}} \left(\frac{-1}{|\mathcal{M}_{\mathbf{s}}| + \alpha_{fd}} + \frac{U_{\mathbf{s}}}{|\mathcal{M}_{\mathbf{s}}| + \alpha_{fd}} P_{\text{suf}(\mathbf{s})} + \frac{\alpha_{fd} + U_{\mathbf{s}}\beta_{fd}}{|\mathcal{M}_{\mathbf{s}}| + \alpha_{fd}} \cdot \frac{\partial P_{\text{suf}(\mathbf{s})}}{\partial \beta_{fd}} \right) \tag{7}$$

[3] Grouping contexts based on depth is sufficient for avoiding the problem of deep contexts being used ineffectively (if the hyper-parameters are set carefully). Making context groups depend also on node fanout further improves compression.

And the gradients with respect to a strength parameter α_{fd} are:

$$\frac{\partial \log P_{\mathbf{s}}}{\partial \alpha_{fd}} = \frac{1}{P_{\mathbf{s}}} \left(\frac{\beta_{fd} - \mathcal{M}_{\mathbf{s}}(x)}{(|\mathcal{M}_{\mathbf{s}}| + \alpha_{fd})^2} + \frac{|\mathcal{M}_{\mathbf{s}}| - U_{\mathbf{s}}\beta_{fd}}{(|\mathcal{M}_{\mathbf{s}}| + \alpha_{fd})^2} \cdot P_{\mathrm{suf}(\mathbf{s})} + \frac{\alpha_{fd} + U_{\mathbf{s}}\beta_{fd}}{|\mathcal{M}_{\mathbf{s}}| + \alpha_{fd}} \cdot \frac{\partial P_{\mathrm{suf}(\mathbf{s})}}{\partial \alpha_{fd}} \right). \tag{8}$$

The quantities $\mathcal{M}_{\mathbf{s}}(x)$, $|\mathcal{M}_{\mathbf{s}}|$ and $U_{\mathbf{s}}$ are defined as in equation (4). These gradients are helpful in two ways: firstly, they can be used to update the hyper-parameters during compression, and secondly, they can be used in an offline search to find the optimal parameter settings for a given sequence. These procedures will be described in sections 6.2 and 6.3.

6.2 Dynamic parameter updates

As the optimal settings of the hyper-parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ depend on the sequence being compressed, it is attractive to adjust them adaptively during compression, rather than setting them to fixed values.

One simple way of implementing such an adaptive mechanism is to make small adjustments to the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ parameter matrices each time a symbol is learned, just before the algorithm regularly updates the symbol counts $\mathcal{M}$. The adjustments add the gradients (7) and (8), scaled by a step size δ, to the hyper-parameters:

$$\alpha_{fd} \leftarrow \alpha_{fd} + \delta \cdot \frac{\partial \log P_{\mathbf{s}}}{\partial \alpha_{fd}} \qquad \text{and} \qquad \beta_{fd} \leftarrow \beta_{fd} + \delta \cdot \frac{\partial \log P_{\mathbf{s}}}{\partial \beta_{fd}}. \tag{9}$$

In our implementation, we used a step size of $\delta = 0.003$. The implementation must ensure that each modified parameter value stays within its permitted range, i.e. $\beta_{fd} \in [0, 1]$ and $\alpha_{fd} \in [-\beta_{fd}, +\infty)$.

6.3 Offline optimisation

Optimal settings of the hyper-parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$ for a given sequence $x_1 \dots x_N$ can be found using e.g. a conjugate gradient optimiser. Offline optimisation is a slow procedure, as it requires compressing the chosen file many times. The output of the offline optimisation is parameter values that are optimal for the chosen file. These values will clearly not be optimal for other files, but may work reasonably well on files that are *similar* to the file they were optimised for. For this reason we've used parameters that were optimised offline on a file from the Canterbury corpus to initialise the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ hyper-parameters in PPM-DP (which uses dynamic parameter updates).

7 Results

The compression effectiveness of our algorithm (PPM-DP), as evaluated on the files of the Canterbury and Calgary corpora, is presented in Table 1. PPM-DP (depth 8) compresses human text better than PPMII, PPMZ, CTW, CSE, LZMA, bzip2 and gzip. The $D_{\max} = 16$ variant of PPM-DP compresses even better. On non-text data (files kennedy.xls, ptt5, sum, geo, obj1 and obj2), LZMA has the strongest compression effectiveness. More results can be found at http://inference.org.uk/compression/ppm-dp/.

Results	gzip	LZMA	bzip2	CSE	CTW	PPMZ	PPMII	N8	N8$^+$	N16$^+$
alice29.txt*	2.850	2.551	2.272	2.192	2.075	2.059	2.033	2.019	2.018	**2.015**
asyoulik.txt	3.120	2.848	2.529	2.493	2.322	2.309	2.308	2.289	2.284	**2.280**
cp.html	2.593	2.478	2.479	2.555	2.307	2.158	2.139	2.140	2.121	**2.113**
fields.c	2.244	2.152	2.180	2.276	1.990	1.896	1.845	1.845	1.820	**1.799**
grammar.lsp	2.653	2.709	2.758	2.750	2.384	2.300	2.268	2.221	2.210	**2.199**
lcet10.txt	2.707	2.233	2.019	1.928	1.832	1.794	1.791	1.787	1.783	**1.773**
plrabn12.txt	3.225	2.746	2.417	2.283	2.185	2.194	2.202	2.178	2.172	**2.171**
xargs.1	3.308	3.369	3.335	3.494	2.962	2.850	2.852	2.782	2.775	**2.771**
kennedy.xls	1.629	**0.409**	1.012	0.891	1.009	1.373	1.168	1.576	1.547	1.519
ptt5 / pic	0.816	**0.618**	0.776	0.772	0.796	0.754	0.757	0.777	0.767	0.768
sum	2.671	**1.982**	2.701	3.024	2.571	2.538	2.327	2.525	2.448	2.399
bib	2.509	2.199	1.975	1.975	1.833	1.718	1.726	1.728	1.715	**1.697**
book1	3.250	2.717	2.420	2.268	2.180	2.188	2.185	2.167	**2.165**	2.166
book2	2.700	2.224	2.062	1.977	1.891	1.839	1.827	1.822	1.819	**1.809**
news	3.063	2.521	2.516	2.525	2.350	2.205	2.188	2.215	2.196	**2.177**
paper1	2.789	2.598	2.492	2.540	2.291	2.212	2.190	2.186	2.179	**2.170**
paper2	2.887	2.655	2.437	2.412	2.229	2.185	2.173	2.164	2.162	**2.158**
progc	2.677	2.532	2.533	2.604	2.337	2.257	2.198	2.218	2.207	**2.192**
procl	1.804	1.666	1.740	1.712	1.647	1.447	1.437	1.487	1.459	**1.415**
progp	1.811	1.671	1.735	1.778	1.679	1.449	1.445	1.544	1.513	**1.432**
trans	1.610	1.420	1.528	1.598	1.443	1.214	1.222	1.301	1.241	**1.195**
geo	5.345	**4.185**	4.447	5.354	4.532	4.578	4.317	4.571	4.383	4.379
obj1	3.837	**3.506**	4.013	4.462	3.721	3.667	3.506	3.658	3.577	3.574
obj2	2.628	**1.991**	2.478	2.711	2.398	2.241	2.160	2.259	2.213	2.173

Table 1: Compression rates of selected compression algorithms on the files of the Canterbury and Calgary corpora. The results are shown in output bits per input byte, with winning entries highlighted in bold. Non-text files are separated from text files with a dotted line.

Our algorithms: N8, N8$^+$ and N16$^+$ are variants of PPM-DP: N8$^+$ uses a maximum context depth of $D_{\max}=8$, and $12_F \times 7_D$ separate parameter pairs. N16$^+$ uses $D_{\max}=16$ and $10_F \times 12_D$ parameter pairs. N8$^+$ and N16$^+$ use dynamic updates as described in section 6.2, and N8 has no dynamic updates. The initial settings of the hyper-parameters were chosen based on an offline optimisation for file alice29.txt, and were hard-wired into the algorithm.

Other algorithms: gzip (Gailly and Adler 1992) uses a variant of the LZ77 algorithm (Ziv and Lempel 1977), bzip2 (Seward 2010) is a compressor based on the Burrows–Wheeler transform (Burrows and Wheeler 1994), LZMA is an algorithm by Pavlov (Pavlov 2011), CTW is the reference implementation of “context tree weighting” by Willems et al. (1993), CSE is an implementation of “compression by substring enumeration” by Dubé and Beaudoin (2010), PPMZ is the PPM variant of Bloom (1998), PPMII is the official implementation of the (depth 16) PPM variant by Shkarin (2001a).

8 Discussion

We proposed a novel data compression algorithm based on the PPM algorithm by Cleary and Witten (1984), taking inspiration from the work on Deplump (Gasthaus and Teh 2010; Bartlett and Wood 2011) and PPMII (Shkarin 2001a, 2002), and insights from Chen and Goodman (1998) and Bunton (1997).

PPM-DP's runtime is comparable to (but slightly slower than) that of other PPM algorithms. Our algorithm uses the same information as other finite-depth PPM implementations (a trie data structure with symbol counts), and therefore requires exactly the same amount of memory.

Limitations. The use of blending and dynamic parameter updates may slow down the algorithm slightly, adding a small (constant) overhead per symbol. In our current implementation, we made no attempts to take computational shortcuts (such as those in Shkarin's PPMII source code), but of course such optimisations can and should be made. Our implementation used floating point representation for the α and β hyper-parameters and their gradients, but faster fixed-point arithmetic could be used instead. For example, one speed-up found in PPMII is the "information inheritance" mechanism (Shkarin 2001a), which can be interpreted as a computationally fast approximation to the *blending* mechanism defined in equation (4). Of course, any changes to the probabilistic model (even when intended as an approximation) will also change the analytic form of the gradients. Such changes should therefore be made with caution.

Future directions. We think it will be beneficial to explore different ways of grouping or disaggregating contexts into separate classes. It may be interesting to construct an unbounded depth version of PPM-DP, which might improve compression effectiveness on large files. To produce a highly efficient implementation of PPM-DP, it may be worthwhile incorporating some of the computational techniques from Shkarin's beautiful source code of PPMII (Shkarin 2006). Finally, it could be beneficial to incorporate PPM-DP's probabilistic model into multi-model ensemble compressors such as PAQ (Mahoney 2000, 2002, 2005).

References

Bartlett, Nicholas and Wood, Frank (2011). Deplump for streaming data. In *Proceedings of the Data Compression Conference*, (edited by James A. Storer and Michael W. Marcellin), 363–372. IEEE Computer Society. ISBN 978-1-61284-279-0. ISSN 1068-0314.

Bloom, Charles (1998). Solving the problems of context modelling. Informally published report. URL http://cbloom.com/papers/ppmz.pdf.

Bunton, Suzanne (1997). Semantically motivated improvements for PPM variants. *The Computer Journal*, **40**(2) 76–93. ISSN 0010-4620.

Burrows, Michael and Wheeler, David John (1994). A block-sorting lossless data compression algorithm. Technical Report SRC Research Report 124, Digital Equipment Corporation, Palo Alto, California.

Chen, Stanley F. and Goodman, Joshua (1998). An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Centre for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, USA.

Cleary, John Gerald and Witten, Ian Hugh (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, **32**(4) 396–402. ISSN 0090-6778.

Cormack, Gordon V. (1993). Dynamic Markov compression (`dmc.c`). A file compressor based on the "dynamic Markov compression" algorithm by Cormack and Horspool (1987). Source code. URL `http://plg.uwaterloo.ca/~ftp/dmc/dmc.c`.

Cormack, Gordon V. and Horspool, R. Nigel S. (1987). Data compression using dynamic Markov modelling. *The Computer Journal*, **30**(6) 541–550.

Dubé, Danny and Beaudoin, Vincent (2010). Lossless data compression via substring enumeration. In *Proceedings of the Data Compression Conference*, (edited by James A. Storer and Michael W. Marcellin), 229–238. IEEE Computer Society. ISBN 978-0-7695-3994-2. ISSN 1068-0314.

Gailly, Jean-loup and Adler, Mark (1992). `gzip`. An open source file compressor based on the `DEFLATE` algorithm by Katz and Burg (1993). Source code. URL `http://www.gnu.org/software/gzip/`.

Gasthaus, Jan and Teh, Yee Whye (2010). Improvements to the Sequence Memoizer. In *Advances in Neural Information Processing Systems 23*, (edited by J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel and A. Culotta), 685–693. ISBN 978-1-61782-380-0.

Gasthaus, Jan, Wood, Frank and Teh, Yee Whye (2010). Lossless compression based on the Sequence Memoizer. In *Proceedings of the Data Compression Conference*, (edited by James A. Storer and Michael W. Marcellin), 337–345. IEEE Computer Society. ISBN 978-0-7695-3994-2. ISSN 1068-0314.

Howard, Paul Glor (1993). *The Design and Analysis of Efficient Lossless Data Compression Systems*. Ph.D. thesis, Department of Computer Science, Brown University, Providence, Rhode Island 02912, USA.

Katz, Phil (1993). `APPNOTE.TXT` — .ZIP file format specification, version 2.0. Distributed by PKWARE Inc. (Milwaukee, Wisconsin, USA). Contains the first published description of the DEFLATE algorithm developed by Katz and Burg (1993).

Katz, Phil and Burg, Steve (1993). `PKZIP`, version 2.04g. Software. Distributed by PKWARE Inc. (Milwaukee, Wisconsin, USA). First implementation of the DEFLATE algorithm, described by Katz (1993).

Mahoney, Matthew Vincent (2000). Fast text compression with neural networks. In *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2000)*, (edited by James N. Etheredge and Bill Z. Manaris), 230–234. AAAI Press. ISBN 978-1-57735-113-9.

Mahoney, Matthew Vincent (2002). The PAQ1 data compression program. Unpublished draft, URL `http://cs.fit.edu/~mmahoney/compression/paq1.pdf`.

Mahoney, Matthew Vincent (2005). Adaptive weighing of context models for lossless data compression. Technical Report CS-2005-16, Department of Computer Science, Florida Institute of Technology, Melbourne, FL, USA.

Moffat, Alistair (1990). Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, **38**(11) 1917–1921. ISSN 0090-6778.

Pavlov, Igor (2003). 7-Zip, version 3.13. Source Code, first published version. URL http://www.7-zip.org/.

Pavlov, Igor (2011). LZMA SDK. Source Code. URL http://www.7-zip.org/sdk.html.

Seward, Julian (2010). `bzip2`, version 1.0.6. A compressor based on the block-sorting transform of Burrows and Wheeler (1994). Source code. URL http://www.bzip.org/.

Shkarin, Dmitry A. (2001a). Повышение эффективности алгоритма PPM (Improving the efficiency of the PPM algorithm). *Проблемы Передачи Информации (Problems of Information Transmission)*, **37**(3) 44–54. ISSN 0555-2923. For an English translation see Shkarin (2001b).

Shkarin, Dmitry A. (2001b). Improving the efficiency of the PPM algorithm. *Problems of Information Transmission*, **37**(3) 226–235. ISSN 1608-3253. Translated from Russian (Shkarin 2001a). See also (Shkarin 2002).

Shkarin, Dmitry A. (2002). PPM: One step to practicality. In *Proceedings of the Data Compression Conference*, (edited by James A. Storer and Martin Cohn), 202–211. IEEE Computer Society. ISBN 978-0-7695-1477-2. ISSN 1068-0314.

Shkarin, Dmitry A. (2006). `ppmdj`. An implementation of PPMII by Shkarin (2001a). Source code. URL http://www.compression.ru/ds/ppmdj.rar.

Steinruecken, Christian (2014). *Lossless Data Compression.* Ph.D. thesis, University of Cambridge.

Teh, Yee Whye (2006). A Bayesian interpretation of interpolated Kneser–Ney. Technical Report TRA2/06, School of Computing, National University of Singapore.

Willems, Frans M. J. (1998). The context-tree weighting method: Extensions. *IEEE Transactions on Information Theory*, **44**(2) 792–798. ISSN 0018-9448.

Willems, Frans M. J., Shtarkov, Yuri M. and Tjalkens, Tjalling J. (1993). Context tree weighting: A sequential universal source coding procedure for FSMX sources. In *International Symposium on Information Theory, Proceedings*, 59. IEEE. ISBN 978-0-7803-0878-7.

Willems, Frans M. J., Shtarkov, Yuri M. and Tjalkens, Tjalling J. (1995). The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, **41**(3) 753–664. ISSN 0018-9448.

Wood, Frank, Archambeau, Cédric, Gasthaus, Jan, James, Lancelot and Teh, Yee Whye (2009). A stochastic memoizer for sequence data. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, (edited by Léon Bottou and Michael L. Littman), volume 382 of *ACM International Conference Proceeding Series*, 1129–1136. ISBN 978-1-60558-516-1.

Ziv, Jacob and Lempel, Abraham (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, **IT-23**(3) 337–343. ISSN 0018-9448.