Compressed Nonparametric Language Modelling

Ehsan Shareghi,♣ Gholamreza Haffari,♣ Trevor Cohn♠
♣ Faculty of Information Technology, Monash University
♠ Computing and Information Systems, The University of Melbourne
first.last@{monash.edu, unimelb.edu.au}

Abstract

Hierarchical Pitman-Yor Process priors are compelling for learning language models, outperforming point-estimate based methods. However, these models remain unpopular due to computational and statistical inference issues, such as memory and time usage, as well as poor mixing of sampler. In this work we propose a novel framework which represents the HPYP model compactly using compressed suffix trees. Then, we develop an efficient approximate inference scheme in this framework that has a much lower memory footprint compared to full HPYP and is fast in the inference time. The experimental results illustrate that our model can be built on significantly larger datasets compared to previous HPYP models, while being several orders of magnitudes smaller, fast for training and inference, and outperforming the perplexity of the state-of-the-art Modified Kneser-Ney count-based LM smoothing by up to 15%.

1 Introduction

Statistical Language Models (LM) are the key components of many tasks in Natural Language Processing, such as Statistical Machine Translation, and Speech Recognition. Conventional LMs are n-gram models which apply the Markov assumption to approximate the probability of a sequence w_1^N as,

$$P(w_1^N) = \prod_{i=1}^{N} P(w_i|w_1^{i-1}) \approx \prod_{i=1}^{N} P(w_i|w_{i-n+1}^{i-1}). \quad (1)$$

Several smoothing techniques have been proposed to address the statistical sparsity issue in computation of each conditional probability term. The widely used smoothing techniques for LM are Kneser-Ney (KN) [Kneser and Ney, 1995], and its extension Modified Kneser-Ney (MKN) [Chen and Goodman, 1999]. The intuition behind KN, MKN and their extensions [Shareghi *et al.*, 2016a] is to adjust the original distribution to assign non-zero probability to unseen or rare events. This is achieved by re-allocating the probability mass in an interpolative procedure via absolute discounting.

It turns out that the Bayesian generalisation of KN family of smoothing is the Hierarchical Pitman-Yor Process (HPYP) LM [Teh, 2006a], which was originally developed for finite-order LM [Teh, 2006b], and was extended as the Sequence Memoizer (SM) [Wood *et al.*, 2011] to model infinite-order LMs. While capturing the long range dependency via HPYP improves the estimation of conditional probabilities, these types of models remain impractical due to several computational and learning challenges, namely large model size (data structure representing the model, and the number of parameters), long training and test time, and poor sampler mixing.

In this paper we address aforementioned issues; inspired by the recent advances in using compressed data structures in LM [Shareghi *et al.*, 2015; 2016b] our model is built on top of a compressed suffix tree (CST) [Ohlebusch *et al.*, 2010]. In the training step, only the CST representation of text is constructed, allowing for a very fast training, while proposing an efficient approximate inference algorithm for the test time. Mixing issue is avoided via careful sampler initialisation and design.

The empirical results show that our proposed approximation of HPYP is richer than KN and MKN, and is much more efficient in learning and inference phase compared to full HPYP. Compared with 10-gram KN and MKN models, our ∞-gram model consistently improves the perplexity by up to 15%. Our compressed framework allows us to train on large collection of text, i.e. $100\times$ larger than the largest dataset used in HPYP LMs [Wood *et al.*, 2011] while having several orders of magnitudes smaller memory footprint and supporting fast and efficient inference.

2 Interpolative Language Models

Conventional interpolative smoothing techniques in LM follow a general form,

$$P(w_i|\mathbf{u}) = \frac{c(\mathbf{u}w_i) - d}{c(\mathbf{u})} + \frac{\gamma(\mathbf{u}, d)}{c(\mathbf{u})} \bar{P}(w_i|\pi(\mathbf{u})),$$

where, $\mathbf{u} = w_{i-n+1}^{i-1}$ is called the context, $\pi(\mathbf{u})$ is $\mathbf{u}$ with its least recent symbol dropped, and c and d are the count and absolute discount, while γ is the mass allocated to the lower level of the interpolation. Interpolative smoothing assumes that $P(w|\mathbf{u})$, the conditional distribution of a word w in the context $\mathbf{u}$, is similar to and hence smoothed by that of suffix

k	u	$P^k_{\text{KN}}(w_i\|\mathbf{u}, d_k)$	$P^k_{\text{HPYP}}(w_i\|\mathbf{u}, \eta^{\mathbf{u}})$
n	u_1	$\frac{c(\mathbf{u}w_i)-d_k}{c(\mathbf{u})}+\frac{d_k N_{1+}(\mathbf{u}\bullet)}{c(\mathbf{u})}P^{k-1}_{\text{KN}}(w_i\|\pi(\mathbf{u}), d_{k-1})$	$\frac{n^{\mathbf{u}}_{w_i}-d^{\mathbf{u}}t^{\mathbf{u}}_{w_i}}{n^{\mathbf{u}}_{\cdot}+\theta^{\mathbf{u}}}+\frac{\theta^{\mathbf{u}}+d^{\mathbf{u}}t^{\mathbf{u}}_{\cdot}}{n^{\mathbf{u}}_{\cdot}+\theta^{\mathbf{u}}}P^{k-1}_{\text{HPYP}}(w_i\|\pi(\mathbf{u}), \eta^{\pi(\mathbf{u})})$
n-1	$\pi(u_1)$	$\frac{N_{1+}(\bullet\,\mathbf{u}w_i)-d_k}{N_{1+}(\bullet\,\mathbf{u}\,\bullet)}+\frac{d_k N_{1+}(\mathbf{u}\bullet)}{N_{1+}(\bullet\,\mathbf{u}\,\bullet)}P^{k-1}_{\text{KN}}(w_i\|\pi(\mathbf{u}), d_{k-1})$	$\frac{n^{\mathbf{u}}_{w_i}-d^{\mathbf{u}}t^{\mathbf{u}}_{w_i}}{n^{\mathbf{u}}_{\cdot}+\theta^{\mathbf{u}}}+\frac{\theta^{\mathbf{u}}+d^{\mathbf{u}}t^{\mathbf{u}}_{\cdot}}{n^{\mathbf{u}}_{\cdot}+\theta^{\mathbf{u}}}P^{k-1}_{\text{HPYP}}(w_i\|\pi(\mathbf{u}), \eta^{\pi(\mathbf{u})})$
...	...	...	...
1	ε	$\frac{N_{1+}(\bullet\,\mathbf{u}w_i)-d_k}{N_{1+}(\bullet\,\mathbf{u}\,\bullet)}+\frac{d_k N_{1+}(\mathbf{u}\bullet)}{N_{1+}(\bullet\,\mathbf{u}\,\bullet)}\frac{1}{\|\sigma\|}$	$\frac{n^{\mathbf{u}}_{w_i}-d^{\mathbf{u}}t^{\mathbf{u}}_{w_i}}{n^{\mathbf{u}}_{\cdot}+\theta^{\mathbf{u}}}+\frac{\theta^{\mathbf{u}}+d^{\mathbf{u}}t^{\mathbf{u}}_{\cdot}}{n^{\mathbf{u}}_{\cdot}+\theta^{\mathbf{u}}}\frac{1}{\|\sigma\|}$

Table 1: One-to-One mapping of interpolative smoothing under KN and HPYP. In P^k_{KN} column : $N_{1+}(\mathbf{u}\bullet) = |\{w : c(\mathbf{u}w) > 0\}|$, and similarly $N_{1+}(\bullet\,\mathbf{u})$ and $N_{1+}(\bullet\,\mathbf{u}\,\bullet)$ are defined. In P^k_{HPYP} column : $n^{\mathbf{u}}_{w_i} = (\mathbf{u}w_i)$ and $n^{\mathbf{u}}_{\cdot} = c(\mathbf{u})$ when $k = n$. Also $n^{\mathbf{u}}_{\cdot} = \sum_{w\in\sigma_{\mathbf{u}}} n^{\mathbf{u}}_w$, and $t^{\mathbf{u}}_{\cdot}$ is defined similarly, and $\eta^{\mathbf{u}} = \{d^{\mathbf{u}}, \theta^{\mathbf{u}}, \{n^{\mathbf{u}}_w, t^{\mathbf{u}}_w\}_{w\in\sigma_{\mathbf{u}}}\}$.

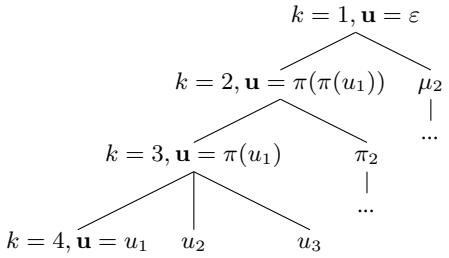


Figure 1: An example of a interpolative LM of depth 3. Moving from leaf node u_1 towards the root, corresponds to moving from the top row towards the bottom row in Table 1. Nodes sharing a parent correspond to identical sequences except for their least recent symbol, for example a partial sequence assignment to the nodes is $\pi(\pi(u_1)) = c$, $\pi_1(u_1) = bc$, $u_1 = abc$, $u_2 = bbc$, $u_3 = dbc$.

of the context $P(w|\pi(\mathbf{u}))$. This recursive smoothing stops at the unigram level where the conditioning context is empty, $\mathbf{u} = \varepsilon$ (see the left panel of Table 1). In what follows, we provide a brief overview of hierarchical Pitmon-Yor Process LM and its relationship with KN.

2.1 Hierarchical Pitman-Yor Process (HPYP) LM

We start by describing the Pitman-Yor process (PYP; [Pitman and Yor, 1997]), used as a prior over LM parameters. PYP(d, θ, H) is a distribution over the space of probability distributions with three parameters: a base distribution H which is the expected value of a draw from a PYP, the concentration parameter $-d < \theta$ which controls the variation of draws from PYP around H, and the discount parameter $0 \leq d < 1$ which controls the heavy-tailness of the sampled distributions. PYP is an appropriate prior for LMs to capture power-law behaviour prevalent in the natural language [Goldwater *et al.*, 2011].

To illustate the use of the PYP prior, we conside as the likelihood a simple unigram LM G, from which the words of a text are generated. The Chinese restaurant process (CRP) is a metaphor that allows generating words from a PYP without directly dealing with the LM G itself by integrating it out. Consider a restaurant where customers are seated on different tables, and each table is served one dish. To make the analogy, the restaurant corresponds to the LM, the customers are the text token needed to be generated, and the dishes are the words. Let t_w denote the number of tables serving the same dish w in the restaurant, n_w denote the total number of customers seated on these tables, and $t_{\cdot} = \sum_w t_w$ and $n_{\cdot} = \sum_w n_w$ to be the total number of tables and customers, respectively. Generating the next word from the LM is done by sending a customer to the restaurant which either (i) sits on an existing table serving the dish w with probability proportional to $n_w - dt_w$, or (2) sits on a new table with probability proportional to $\theta + dt_{\cdot}$ and orders a dish from the base distribution H. The probability of the next word w is thus

$$P(w|\eta) = \frac{n_w - dt_w}{n_{\cdot} + \theta} + \frac{\theta + dt_{\cdot}}{n_{\cdot} + \theta} P(w|H).$$

where $\eta = \{d, \theta, \{n_w, t_w\}_{w\in\sigma}\}$, and σ is the vocabulary. Note that $\{n_w, t_w\}_{w\in\sigma}$ form the sufficient statistics for generating the next word.

In a HPYP LM, the distribution $G^{\mathbf{u}}$ of words following a context $\mathbf{u}$ has a PYP$(d^{\mathbf{u}}, \theta^{\mathbf{u}}, G^{\pi(\mathbf{u})})$ prior,

$$G^{\mathbf{u}} \mid d^{\mathbf{u}}, \theta^{\mathbf{u}}, G^{\pi(\mathbf{u})} \sim \text{PYP}(d^{\mathbf{u}}, \theta^{\mathbf{u}}, G^{\pi(\mathbf{u})}).$$

where the base distribution $G^{\pi(\mathbf{u})}$ itself has a PYP prior. This induces a hierarchy among these context-conditioned distributions (see Figure 1) tying the base distribution of each node of the hierarchy to the distribution of its parent. Note that $\mathbf{u}$ refers to both a context (a sequence of words) and its corresponding node in the HPYP tree. The distribution at the root of the hierarchy G^{ε} corresponds to the empty context ε:

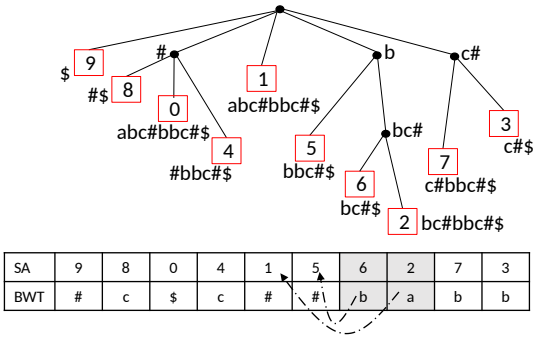
$$G^{\varepsilon} \mid d^{\varepsilon}, \theta^{\varepsilon}, \mathcal{U} \sim \text{PYP}(d^{\varepsilon}, \theta^{\varepsilon}, \mathcal{U})$$

where the base distribution $\mathcal{U}$ is the uniform distribution over the vocabulary σ.

Having a HPYP LM, the next word is generated by integrating out all of the distributions corresponding to the nodes of the tree. This is achieved by hierarchical Chinese restaurant process (HCRP), whereby a customer is sent to a restaurant of a context from which a word needs to be generated. In case the customer is seated on a new table, a new customer is sent to the parent restaurant for ordering the dish. The number of customers sent to the parent is orchestrated via the concentration and discount parameters. The following constraints hold for $\{n^{\mathbf{u}}_w, t^{\mathbf{u}}_w\}_{w\in\sigma_{\mathbf{u}}}$ across the tree nodes:

$$\forall w \in \sigma_{\mathbf{u}} : \quad 0 < t^{\mathbf{u}}_w \leq n^{\mathbf{u}}_w \tag{2}$$

$$\forall w \in \sigma_{\mathbf{u}} : \quad n^{\mathbf{u}}_w = \sum_{\boldsymbol{\psi}\in\text{children}(\mathbf{u})} t^{\boldsymbol{\psi}}_w \tag{3}$$



SA	9	8	0	4	1	5	6	2	7	3
BWT	#	c	$	c	#	#	b	a	b	b

Figure 2: (top) Suffix Tree, (bottom) Suffix Array and Burrows-Wheeler Transformation for "$\#abc\#bbc\#\$$". In (top), labels correspond to concatenation of edge labels from the root to each node and digits stored on leaves and in SA, (bottom), correspond to the starting index of the suffixes in text.

where $\sigma_{\mathbf{u}}$ is the subset of σ observed after the context $\mathbf{u}$. Tying the distributions as a hierarchy simulates the smoothing process by adjusting the base distributions of each level recursively. Given $\{\eta^{\mathbf{u}}\}^{\mathbf{u}\in\text{HPYP}}$, $\mathbf{u} \in$ HPYP denoting all nodes in HPYP, the predictive probability of a word w in a context $\mathbf{u}$ is computed recursively as $P_{\text{HPYP}}(w|\mathbf{u})$ in Table 1.

Note the close similarity in formulation of KN smoothing and the HPYP as illustrated in Table 1; indeed we recover KN smoothing from HPYP by assuming $\{\theta^{\mathbf{u}} = 0\}^{\mathbf{u}\in\text{HPYP}}$ and $\{t^{\mathbf{u}}_w = 1\}^{\mathbf{u}\in\text{HPYP}}_{w\in\sigma_{\mathbf{u}}}$. In what follows, we exploit this property to compactly represent the HPYP LM.

3 Compressed HPYP LM

The child-parent relationship between distributions in HPYP forms a context tree which can be represented as a suffix trie, or more compactly as a suffix tree [Wood *et al.*, 2011]. In practice suffix trees require at best $20|\mathcal{T}|$ bytes of space, where $\mathcal{T}$ denotes the text, making them impractical for anything but small data.

Hence, even storing the structure of the HPYP model without storing parameters $\{\eta^{\mathbf{u}}\}^{\mathbf{u}\in\text{HPYP}}$ is impractical for large datasets. This was possibly the case given that the largest dataset that HPYP LMs were built on small corpora including only a few million words [Teh, 2006b; Gasthaus *et al.*, 2010; Wood *et al.*, 2011]. However, with the availability of datasets of orders of magnitudes larger size [Buck *et al.*, 2014; Parker *et al.*, 2011], it is crucial to improve the scalability of HPYP LMs. In the following, we briefly introduce a set of compressed data structures and operations, and then illustrate how a HPYP LM can be made scalable using these tools.

3.1 Compressed Suffix Tree

A Suffix Tree (ST) [Weiner, 1973] of a string $\mathcal{T}$ with alphabet σ is a tree of $|\mathcal{T}| + 1$ leaves, where a path from the root to a leaf corresponds to a suffix of $\mathcal{T}$. Each leaf holds a number indicating the staring position of the suffix in $\mathcal{T}$ while leaves are lexicographically ordered, see Figure 2(top). The search for any sequence $\mathbf{u}$ in $\mathcal{T}$ corresponds to finding the node v in ST such that $\mathbf{u}$ is a prefix of the concatenation of the path labels from the root to v. While suffix trees offer $\mathcal{O}(|\mathbf{u}|)$ search complexity, in practice they require $20|\mathcal{T}|$ bytes.

DEFINITION	OPERATION	COMPLEXITY
search($\mathbf{u}$)	*bw-search*	$\mathcal{O}(\|\mathbf{u}\| \log \|\sigma\|)$
c($\mathbf{u}$)	*size*	$\mathcal{O}(1)$
$N_{1+}(\mathbf{u}\bullet)$	*deg*	$\mathcal{O}(1)$
$N_{1+}(\bullet\mathbf{u})$	*int-sym*	$\mathcal{O}(N_{1+}(\bullet\mathbf{u}) \log \|\sigma\|)$
$N_{1+}(\bullet\mathbf{u}\bullet)$	*deg+int-sym*	$N_{1+}(\bullet\mathbf{u})\mathcal{O}(1)+\mathcal{O}(N_{1+}(\bullet\mathbf{u}) \log \|\sigma\|)$

Table 2: Key CST operations, definitions, and time complexities. The four operations at the bottom assume the node matching $\mathbf{u}$ is given via a *backward-search (bw-search)*.

A Suffix Array (SA) [Manber and Myers, 1993] of $\mathcal{T}$ is an array of sorted suffixes of $\mathcal{T}$, where SA$[i]$ holds a same number as i-th leaf in ST, see Figure 2(bottom). Search in SA translates to binary search to find the corresponding range that spans over all substrings that have $\mathbf{u}$ as their prefix, and is $\mathcal{O}(|\mathbf{u}| \log |\mathcal{T}|)$. Constructing SA takes 4-8$|\mathcal{T}|$ bytes in practice, which compared to $|\mathcal{T}| \log |\sigma|$ bits required to store $\mathcal{T}$ makes both ST and SA impractical to use for large data.

A Compressed Suffix Array (CSA) exploits text compressibility and provides the same functionality as SA but in space equal to bzip2 compressed $\mathcal{T}$ in practice. We use the FM-Index [Ferragina *et al.*, 2008] that utilizes the text compressibility by using the Burrows-Wheeler transformation (BWT) [Burrows and Wheeler, 1994] of the text, which is defined as BWT$[i]$ = [SA$[i] - 1$ mod $|\mathcal{T}|]$ as illustrated in Figure 2(bottom). Searching for a sequence in BWT is done in reverse order (called *backward-search*) and requires $\mathcal{O}(|\mathbf{u}| \log |\sigma|)$. Similarly, a Compressed Suffix Tree (CST) simulates ST and is built on CSA by storing extra bits to store the shape of the tree and path labels [Ohlebusch *et al.*, 2010]. For details see [Shareghi *et al.*, 2015].

Table 2 illustrates the key operations on CSA and CST along with their complexities. The *backward-search* looks-up the CSA span covering the given sequence and returns the $[lb, rb]$ of the node v matching it, and the *size* operation counts the number of leaves of the CST subtree rooted at v using the length of the returned range. The *degree* operation returns the number of types completing a sequence to its right. The most expensive operation is the *interval-symbols*, which for a context $\pi(\mathbf{u})$ returns its corresponding set of children and each child's corresponding node's range using the BWT [Schnattinger *et al.*, 2010]. Figure 2(bottom) shows the interval symbol procedure for finding the children of "$\pi(\mathbf{u}) = bc$": First a search is done to find the corresponding range for "bc" in SA, highlighted in gray, then the corresponding cells on BWT are looked up to find $\{b, a\}$ as the two possible completions of "bc" to its left, and then their corresponding nodes are located efficiently, as illustrated by dashed arrows.

3.2 Compressed HPYP LM

We make use of a CST to represent HPYP LM compactly, resulting in a model which is less than the size of the text itself. This is inspired by the use of CSTs to represent KN family of smoothing, and the fact that the structure of the hierarchy representing both models are exactly the same. The basic idea for compressed KN-smoothed LMs is to extract the required counts directly from a CST

representation of the text on-the-fly [Shareghi *et al.*, 2015; 2016b]; Table 2 covers the required quantities together with the CST operations and their time complexities.

The avid readers might notice that the key difference between KN and HPYP is in the $\{n^{\mathbf{u}}_w, t^{\mathbf{u}}_w\}^{\mathbf{u}\in \text{HPYP}}$. While it is possible to store vectors $\{n^{\mathbf{u}}_w, t^{\mathbf{u}}_w\}_{w\in\sigma_{\mathbf{u}}}$ for each node $\mathbf{u}$ of HPYP, it adds a significant load to the memory usage. This is one of the key computational issues of the inference approach taken in sequence memoizer [Wood *et al.*, 2011], which samples these vectors in the training time and stores them before using them in testing. We address the issue by moving the sampling to the test time, hence skipping the need to store these samples.

To complete our model description, we need to cover the discount and concentration parameters. In our model, the discount parameters are set to Kneser-Ney discounts and tied based on the context size $|\mathbf{u}|$, while each distribution uses its own separate concentration parameter. Our decision for fixing the discount parameters was to avoid the cost of sampling them during the inference. Also, the range for the discount parameters are very fine-grained making the gain in sampling them very negligible[1]. We show in the inference section how fixing the discounts allows us to develop an efficient sampler.

4 Fast Approximate Inference for HPYP LM

The inference in a HPYP LM translates into computing the predictive probability of a word w, in the context $\mathbf{u}$. This corresponds to integrating out all the latent prior distributions and is defined as the following intractable integral,

$$P(w|\mathbf{u}) = \int P(w|\mathbf{u},\eta)P(\eta)d(\eta) \tag{4}$$

and is approximated using samples for η. Here $\eta = \{\eta^{\mathbf{u}}\}^{\mathbf{u}\in\text{HPYP}}$, and $P(w|\mathbf{u},\eta)$ is defined as in Table 1 (right panel). While there are different approaches to generate samples, they are designed to run in the training phase and require explicit storage of sampled quantities. This amounts to a significant memory load which we avoid by skipping it. Consequently, it is required to generate the samples in the test phase, but the existing sampling algorithms [Gasthaus and Teh, 2010] generate samples across all nodes in HPYP which is too slow to fit our purpose. Instead, we present a novel sampler, designed for the query phase of LM, which results in a much lower memory usage, is fast, and avoids mixing issues inherited in the full samplers.

Let us assume the path from the root to the node matching the context $\mathbf{u}$ of a given query "$\mathbf{u}w_i$" and denote the set of distributions along the path by γ^+, and all the other nodes of the HPYP by γ^-. For example, for the query $P(w=c|\mathbf{u}=ab)$, $\gamma^+ = \{G^{ab}, G^b, G^\varepsilon\}$ and $\gamma^- = \{G^{\mathbf{u}}|\mathbf{u}\in\text{HPYP}\wedge G^{\mathbf{u}}\notin\gamma^+\}$. Then, the three computational solutions involved in our proposed approximate inference scheme are:

Branch Sampling While a full HPYP sampler, i.e. in SM [Wood *et al.*, 2011], samples on $\gamma^+\cup\gamma^-$, in here only the γ^+ distributions and only the type matching the query word w_i are selected for sampling, fixing γ^- at their initialization which is KN. This means at any given state of sampling, we have $\{t^{\mathbf{u}}_w = 1\}^{\mathbf{u}\in\gamma^-}_{w\in\sigma_{\mathbf{u}}}$ and allows for a fast inference in the test time while reducing the size of the sampling space exponentially hence reducing the risk of poor mixing.

Forgetting Samples Samples on γ^+ are generated during the test phase, used for approximating the predictive probability and then forgotten immediately. In this process, for any given query the state of HPYP will be set to KN at the beginning of the sampling process. This keeps the memory usage of the training and inference phase close, and roughly matching the size of the compressed text.

Range Shrinking The key quantities in the sampling phase are $t^{\mathbf{u}}_w$. In practice, the range $0 < t^{\mathbf{u}}_w \leq n^{\mathbf{u}}_w$ can potentially be very large, making the sampling very slow. Instead, we follow a non-uniform sampling by shrinking the range to $1 \leq t^{\mathbf{u}}_w \leq \min\{M, n^{\mathbf{u}}_w\}$ (Here $M = 10$). The motivation here is based on the key difference between KN and MKN which is mainly in the discount range. In MKN, which typically outperforms KN, discounts are larger than 1 and our empirical analysis on several datasizes and languages illustrate the range in practice is $[0, 3]$.[2] This effect to some degree is replicated in HPYP when the discounts $0 \leq d^{\mathbf{u}} < 1$ are multiplied by $t^{\mathbf{u}}_w$. Shrinking the sampling range keeps the HPYP distributions at each level of HPYP close to their corresponding KN (and MKN) counterparts, while the concentration parameter allows the distributions to have more flexibility in capturing the desired distribution.

The first two components allow fast inference while keeping the memory usage of our approach to be several orders of magnitudes smaller than the SM, hence making our approach computationally practical for large data regime. The third component seeks to take advantage of the best component of MKN smoothing, while avoiding the mixing issue that occurs for the infinite HPYP case. In the following two subsections, we provide the statistical underpinning of the sampling, and show how it can be done under CST mechanics.

4.1 Sampling

We sample using the joint distribution P($\{\eta^{\mathbf{u}}\}^{\mathbf{u}\in\text{HPYP}}$),

$$\prod_w H(.)^{t^{\varepsilon}_w} \prod_{\mathbf{u}} \left(\frac{(\theta^{\mathbf{u}}|d^{\mathbf{u}})_{t^{\mathbf{u}}_{\cdot}}}{(\theta^{\mathbf{u}}|1)_{n^{\mathbf{u}}_{\cdot}}} \prod_w S_{d^{\mathbf{u}}}(n^{\mathbf{u}}_w, t^{\mathbf{u}}_w) \right) \tag{5}$$

where $(a|b)_c$ is the Pochhammer[3] symbol, and $S_d(n,t)$ is the generalized Stirling number of kind $(-1,-d,0)$ [Hsu and Shiue, 1998].

The joint distribution in eqn. 5 allows efficient sampling for $t^{\mathbf{u}}_w$ and $n^{\mathbf{u}}_w$ in the hierarchy, starting from the data level and going up in the hierarchy. The only expensive computation is for the Stirling numbers which are cached as KN discounts are used. We use the exact recursive formulation of Stirling numbers [Buntine and Hutter, 2012] and switch to asymptotic approximation[4] when t or n are large, i.e. ≥ 8000.

[1] Our analysis shows no improvements of perplexities when discounts were sampled, while it made the inference step slower.

[2] In theory, the MKN discounts can be as large as $c(\mathbf{u}w_i)$.

[3] $(a|b)_c = a(a+1\times b)...(a+(c-1)\times b)$

[4] Using the Stirling's approximation for factorials.

Algorithm 1 Gibbs Sampler for $\eta \in \gamma^+$

function SAMPLER($w, k, n, \gamma^+, \vec{S}, M$)
  $\mathbf{u}^* \leftarrow \gamma^+_k$, $\pi(\mathbf{u}^*) \leftarrow \gamma^+_{k-1}$
  $t^{\mathbf{u}^*}_w \leftarrow 0$, $n^{\mathbf{u}^*}_w \leftarrow n$
  $\vec{Q} \leftarrow$ null
  if $n^{\mathbf{u}^*}_w \neq 1$ **then**
    while $t^{\mathbf{u}^*}_w \leq \min\{M, n^{\mathbf{u}^*}_w\}$ **do**
      if $\mathbf{u}^* \neq \varepsilon$ **then**
        $F \propto P(t^{\mathbf{u}^*}_w|...)$ ▷ eqn.7
      else
        $F \propto P(t^{\varepsilon}_w|...)$ ▷ eqn.8
      $\vec{Q} \leftarrow (F, t^{\mathbf{u}^*}_w)$
      $t^{\mathbf{u}^*}_w \leftarrow t^{\mathbf{u}^*}_w + 1$
    $t^{\mathbf{u}^*}_w \leftarrow$ sample_from($\vec{Q}$)
  else
    $t^{\mathbf{u}^*}_w \leftarrow 1$
  $\theta^{\mathbf{u}^*} \leftarrow$ sample $\theta^{\mathbf{u}}$
  $\vec{S} \leftarrow (\mathbf{u}^*, n^{\mathbf{u}^*}_w, t^{\mathbf{u}^*}_w, \theta^{\mathbf{u}^*})$
  if $\mathbf{u}^* \neq \varepsilon$ **then**
    $n^{\pi(\mathbf{u}^*)}_w \leftarrow$ update($n^{\pi(\mathbf{u}^*)}_w, t^{\mathbf{u}^*}_w$) ▷ eqn.6
    SAMPLER($w, k-1, n^{\pi(\mathbf{u}^*)}_w, \gamma^+, \vec{S}, M$)
  return($\vec{S}$)

For each $G^{\mathbf{u}} \in \gamma^+$, except the leaf level[5], the $n^{\mathbf{u}}_w$'s will be sampled jointly as $t^{\psi(\mathbf{u})}_w$'s are sampled, where $\psi(\mathbf{u}) \in$ children($\mathbf{u}$). Starting from the leaf level of the hierarchy, the $n^{\mathbf{u}}_w$'s are read from the data, hence fixed and $t^{\mathbf{u}}_w$'s are sampled while satisfying the constraints in eqn.2, and eqn.3. Given a sampled $t^{\mathbf{u}^*}_w$ at the leaf level $\mathbf{u}^*$, the $n^{\pi(\mathbf{u}^*)}_w$ is updated as,

$$n^{\pi(\mathbf{u}^*)}_w = t^{\mathbf{u}^*}_w + \sum_{\psi \in \text{children}(\pi(\mathbf{u}^*)) \wedge \psi \neq \mathbf{u}^*} t^{\psi}_w. \quad (6)$$

The conditional probability of the sampled $t^{\mathbf{u}*}_w$ from eqn. 5 for the *non-root* levels while fixing all the independent variables, $P(t^{\mathbf{u}^*}_w|...)$, is proportional to,

$$\frac{(\theta^{\mathbf{u}^*}|d^{\mathbf{u}^*})_{t^{\mathbf{u}^*}_\cdot}}{(\theta_{\pi(\mathbf{u}^*)}|1)_{\sum_{\psi \in \text{children}(\pi(\mathbf{u}^*))} t^{\psi}_\cdot}} S_{d^{\mathbf{u}^*}}(n^{\mathbf{u}^*}_w, t^{\mathbf{u}^*}_w) S_{d^{\pi(\mathbf{u}^*)}}(n^{\pi(\mathbf{u}^*)}_w, t^{\pi(\mathbf{u}^*)}_w) \quad (7)$$

where $t^{\mathbf{u}^*}_\cdot = t^{\mathbf{u}^*}_w + \sum_{v \neq w} t^{\mathbf{u}^*}_v$, and for the *root* level,

$$P(t^{\varepsilon}_w|...) \propto H(.)^{t^{\varepsilon}_w}(\theta^{\varepsilon}|d^{\varepsilon})_{t^{\varepsilon}_\cdot} S_{d^{\varepsilon}}(n^{\varepsilon}_w, t^{\varepsilon}_w). \quad (8)$$

Given sampled $t^{\mathbf{u}^*}_w, n^{\mathbf{u}^*}_w$ for a context $\mathbf{u}$, the concentration parameter $\theta^{\mathbf{u}^*}$ is sampled via auxiliary variables [Teh *et al.*, 2012] using a Gamma(a,b) prior. Algorithm 1 illustrates the sampling procedure for collecting a single set of samples along γ^+. The algorithm starts from the leaf level and moves up on the γ^+ branch, sampling $t^{\mathbf{u}}_w$ and $n^{\pi(\mathbf{u})}_w$ jointly. The index k denotes the level on the extracted branch, and matches the k in Table1. Given a query, this process is repeated multiple times along the γ^+ branch.

[5]The leaf level is where the data is observed (first row of Table 1).

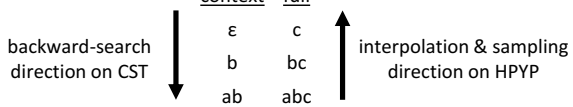


Figure 3: Direction of search, interpolation, and sampling for *"abc"*.

QUANTITY	COMPLEXITY
$t^{\mathbf{u}^*}_\cdot$	$\mathcal{O}(1)$
$\sum_{\psi \in \text{children}(\pi(\mathbf{u}^*))} t^{\psi}_\cdot$	$N_{1+}(\bullet\pi(\mathbf{u}^*))\mathcal{O}(1) + \mathcal{O}(N_{1+}(\bullet\pi(\mathbf{u}^*))\log\lvert\sigma\rvert)$
$n^{\pi(\mathbf{u}^*)}_w$	$N_{1+}(\bullet\pi(\mathbf{u}^*)w) + \mathcal{O}(N_{1+}(\bullet\pi(\mathbf{u}^*)w)\log\lvert\sigma\rvert)$

Table 3: Complexities of computing critical sampling quantities.

4.2 Sampling under CST mechanics

Sampling under CST involves two passes in the opposite directions, see Figure 3. Given a query, one pass starts from the last word of the query and grows the pattern to its left one word at a time. This pass collects all the required nodes in γ^+, and identifies the required fragmentations, while allowing to reuse spans during the backward-search, instead of operating a fresh search over the full CST span. Once all the required nodes are extracted, a second pass in the opposite direction on the γ^+ branch, samples t, n, and θ. The core of sampler relies on eqn. 7 and eqn. 8, which we describe in below. Given a sampled $t^{\mathbf{u}^*}_w$ for a context $\mathbf{u}^*$ and word w, $t^{\mathbf{u}^*}_\cdot$ is defined as $(N_{1+}(\mathbf{u}^* \bullet) - 1) + (t^{\mathbf{u}^*}_w)$ which assumes a table per word type for all the words occurring after $\mathbf{u}$ except for the word w for which $t^{\mathbf{u}^*}_w$ is sampled. This translates into a *degree*($\mathbf{u}$) call to compute $N_{1+}(\mathbf{u}^* \bullet)$ and is done in constant time. Computing the other main quantities is more expensive and involves *interval-symbols* operation. The children nodes of $\mathbf{u}^*$, and $\pi(\mathbf{u}^*)w$ are extracted via *interval-symbols*. Then, their table and count statistics are computed,

$$\sum_{\psi \in \text{children}(\pi(\mathbf{u}^*))} t^{\psi}_\cdot = t^{\mathbf{u}^*}_\cdot + \sum_{\psi \in \text{children}(\pi(\mathbf{u}^*)) \wedge \psi \neq \mathbf{u}^*} t^{\psi}_\cdot$$

$$n^{\pi(\mathbf{u}^*)}_w = t^{\mathbf{u}^*}_w + \sum_{\psi \in \text{children}(\pi(\mathbf{u}^*)w) \wedge \psi \neq u^*} t^{\psi}_w,$$

where for each ψ, except for $\mathbf{u}^*$, a *degree* operation is called, and $t^{\mathbf{u}^*}_\cdot$ is computed as mentioned before. Table 3 illustrates the complexity of these computations.

5 Experiments

We report the perplexity of KN, MKN, SM, and our approach CN using the Finnish (FI), Spanish (ES), German (DE), English (EN), French (FR), portions of the Europarl v7 [Koehn, 2005] corpus, as well as 250MiB, 500MiB, 1,2,4, and 8GiB chunks of English Common Crawl corpus [Buck *et al.*, 2014]. The data was tokenized, sentence split, and the XML markup discarded. As test sets, we used newstest-2014 for all languages except Spanish, for which we used newstest-2013. To

	tokens (M)		PERPLEXITY			
			n=10		n=∞	
	TRAIN	TEST	KN	MKN	SM	CN
EU-DE	54.93	0.06	1810	1694	1598	1543
EU-FI	40.47	0.02	5570	5344	4833	4756
EU-FR	66.79	0.08	1328	1191	1090	1048
EU-ES	62.06	0.07	444	416	440	377
EU-EN	61.30	0.07	921	844	806	725
125MiB	32.52	0.07	333	329	328	289
250MiB	65.01	0.07	299	295	300	283
1GiB	201.52	0.07	246	242	251	224
2GiB	403.47	0.07	223	219	—	209
4GiB	807.71	0.07	204	200	—	190
8GiB	1617.27	0.07	184	181	—	174

Table 4: Data statistics, and perplexities of Kneser-Ney (KN), Modified Kneser-Ney (MKN), Sequence Memoizer (SM), and Compressed Nonparametric (CN) on different datasets. The empty cells for 2,4,8 GiB are due to SM exceeding the 180GiB memory budget.

avoid the effect of differences in handling Out-of-Vocabulary words in measuring the perplexities, we used a closed vocabulary setup. To measure the KN and MKN perplexities we used the SRILM [Stolcke, 2002] toolkit. And to verify the comparability, we forced the KN assumptions on our model and closely matched (difference ≤ 1) the perplexity numbers reported by SRILM. For benchmarking the memory and time usage of CN against SM, we used the English language datasets varying in size, from 125MiB to 8GiB chunks of Common Crawl. All experiments are done on a single core on Intel Xeon E5-2667 3.2GHz and 180GiB of RAM.

Perplexity As illustrated in Table 4, our approach (CN) consistently outperforms MKN perplexities by a margin of up to 15%. To test against full HPYP inference, we compared against the available implementation of SM.[6] Although CN is initialised by KN and samples from the conditional $P(\gamma^+|\gamma^-)$, it is consistently better than SM. We speculate this is due to poor mixing of SM over the full sampling space involving HPYP tree nodes, and proper mixing of our fast inference method over the smaller sampling space involving only nodes on a single branch γ^+. The difference between perplexities across multiple runs of our model were negligible. Comparing SM with KN and MKN reveals a surprising result on some datasets: SM does worse, or is only marginally better regardless of the number of burn-in, or samples.[7]

Memory and Time As demonstrated in Figure 4, the memory used by SM is several orders of magnitudes larger than the size of the text and the size of our model in both training and test (query). For instance, on 250MiB dataset CN used (297MiB,108MiB) in training and test, compared with

[6]https://github.com/jgasthaus/libPLUMP

[7]This doesn't verify the comparison reported in [Wood *et al.*, 2011]. We noticed a critical decision in their experimental setup: while setting a threshold to replace low frequency words with a single token is a popular approach in text processing, in the KN and MKN LM this will cause a range of discount parameters to be zero, eliminating the effect of smoothing and making KN and MKN perform worse than their full potential.

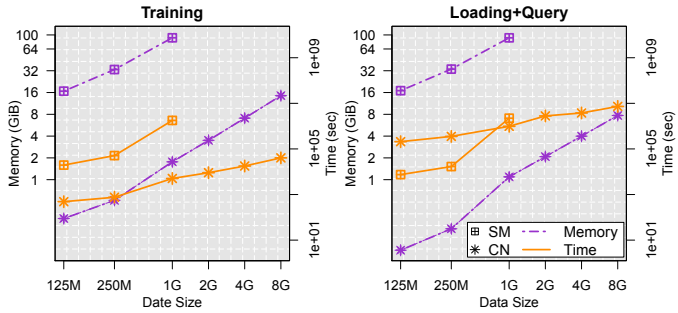


Figure 4: Time (right Y-axis) and Memory (left Y-axis) usage of SM and CN in training and query phases on various data sizes (X-axis). SM could not be run on $2 \leq$ GiB due to our memory budget.

(16GiB,16GiB) of SM when it stores only 1 set of HPYP samples. This made it impossible for us to run SM on larger ($\geq$ 2GiB) datasets, or with more samples. In terms of time usage in the training step, our approach is several times faster, i.e., (40×,67×) on (125MiB,250MiB) datasets. Noting that SM tends to get slower on larger datasizes, taking more than 14 days to train on 1GiB dataset, while we only required less than 2 hours. In the test time we are on average around 24× slower noting that SM tends to get slower on larger datasizes, i.e. from 27× faster on 125MiB to 21× on 250MiB. Inference only with 5 samples along each branch and no burn-in, affected the perplexities of our approach up to 2% while making the average test speed only 4× slower than SM. Our approach on "load+query" carries roughly a similar pattern excluding the load time. On 1GiB, our query is only 1.8× slower than SM, and we are 2.3× faster on load+query. A significant result which is due to smaller model size, and efficient inference mechanism of our CST-based framework.

6 Conclusion

In this paper we proposed a framework based on compressed suffix trees to represent infinite-order *hierarchical* Bayesian language models compactly, while developing a fast and memory-efficient approximate inference scheme. Compared with the existing HPYP LMs our approach has several orders of magnitudes lower memory footprint allowing us to apply it on (100×) larger data sizes than the largest data used by HPYP LM. This is achieved by avoiding potential mixing issues, while consistently outperforming the Kneser-Ney family of smoothings by a significant margin.

As our future work, we would like to speedup the inference via approximating Stirling numbers using a separate model to avoid its expensive recursion cost during sampling, as well as exploring continuous space approximations of HPYP.

Acknowledgments

This research was supported by the National ICT Australia (NICTA). The first author would like to thank Wray Buntine for fruitful discussions about sampling in HPYP, and Philip Chan for the support to run the experiments on Monash Advanced Research Computing Hybrid (MonARCH) servers.

References

[Buck *et al.*, 2014] Christian Buck, Kenneth Heafield, and Bas van Ooyen. N-gram counts and language models from the common crawl. In *Proceedings of the Language Resources and Evaluation Conference*, 2014.

[Buntine and Hutter, 2012] Wray Buntine and Marcus Hutter. A bayesian view of the Poisson-Dirichlet process. *arXiv preprint arXiv:1007.0296*, 2012.

[Burrows and Wheeler, 1994] Michael Burrows and David Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation Systems Research Center, 1994.

[Chen and Goodman, 1999] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[Ferragina *et al.*, 2008] Paolo Ferragina, Rodrigo González, Gonzalo Navarro, and Rossano Venturini. Compressed text indexes: From theory to practice. *ACM J. of Exp. Algorithmics*, 13, 2008.

[Gasthaus and Teh, 2010] Jan Gasthaus and Yee W. Teh. Improvements to the sequence memoizer. In *Advances in Neural Information Processing Systems 23*, pages 685–693, 2010.

[Gasthaus *et al.*, 2010] Jan Gasthaus, Frank Wood, and Yee Whye Teh. Lossless compression based on the sequence memoizer. In *2010 Data Compression Conference (DCC 2010)*, pages 337–345, 2010.

[Goldwater *et al.*, 2011] Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. Producing power-law distributions and damping word frequencies with two-stage language models. *Journal of Machine Learning Research*, 12(Jul):2335–2382, 2011.

[Hsu and Shiue, 1998] Leetsch C Hsu and Peter Jau-Shyong Shiue. A unified approach to generalized stirling numbers. *Advances in Applied Mathematics*, 20(3):366–384, 1998.

[Kneser and Ney, 1995] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 181–184, 1995.

[Koehn, 2005] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of the Machine Translation summit*, 2005.

[Manber and Myers, 1993] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[Ohlebusch *et al.*, 2010] Enno Ohlebusch, Johannes Fischer, and Simon Gog. CST++. In *Proceedings of the International Symposium on String Processing and Information Retrieval*, 2010.

[Parker *et al.*, 2011] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition. *Linguistic Data Consortium*, (LDC2011T07), 2011.

[Pitman and Yor, 1997] Jim Pitman and Marc Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, pages 855–900, 1997.

[Schnattinger *et al.*, 2010] Thomas Schnattinger, Enno Ohlebusch, and Simon Gog. Bidirectional search in a string with wavelet trees. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, 2010.

[Shareghi *et al.*, 2015] Ehsan Shareghi, Matthias Petri, Gholamreza Haffari, and Trevor Cohn. Compact, efficient and unlimited capacity: Language modeling with compressed suffix trees. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

[Shareghi *et al.*, 2016a] Ehsan Shareghi, Trevor Cohn, and Gholamreza Haffari. Richer interpolative smoothing based on modified kneser-ney language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[Shareghi *et al.*, 2016b] Ehsan Shareghi, Matthias Petri, Gholamreza Haffari, and Trevor Cohn. Fast, small and exact: Infinite-order language modelling with compressed suffix trees. *Transactions of the Association for Computational Linguistics*, 4:477–490, 2016.

[Stolcke, 2002] Andreas Stolcke. SRILM–an extensible language modeling toolkit. In *Proceedings of the International Conference of Spoken Language Processing*, 2002.

[Teh *et al.*, 2012] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the american statistical association*, 2012.

[Teh, 2006a] Yee Whye Teh. A Bayesian interpretation of interpolated Kneser-Ney. Technical report, NUS School of Computing, 2006.

[Teh, 2006b] Yee Whye Teh. A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2006.

[Weiner, 1973] Peter Weiner. Linear pattern matching algorithms. In *Proceedings of the Annual Symposium Switching and Automata Theory*, 1973.

[Wood *et al.*, 2011] Frank Wood, Jan Gasthaus, Cédric Archambeau, Lancelot James, and Yee Whye Teh. The sequence memoizer. *Communications of the ACM*, 54(2):91–98, 2011.