Hierarchical Bayesian Nonparametric Models for Power-Law Sequences

Jan Alexander Gasthaus

Gatsby Computational Neuroscience Unit
University College London

Thesis

submitted for the degree of

Doctor of Philosophy

2020

Declaration

I, Jan Alexander Gasthaus, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Jan Alexander Gasthaus

Abstract

Sequence data that exhibits power-law behavior in its marginal and conditional distributions arises frequently from natural processes, with natural language text being a prominent example. We study probabilistic models for such sequences based on a hierarchical non-parametric Bayesian prior, develop inference and learning procedures for making these models useful in practice and applicable to large, real-world data sets, and empirically demonstrate their excellent predictive performance. In particular, we consider models based on the infinite-depth variant of the hierarchical Pitman-Yor process (HPYP) language model [Teh, 2006b] known as the *Sequence Memoizer*, as well as Sequence Memoizer-based cache language models and hybrid models combining the HPYP with neural language models. We empirically demonstrate that these models perform well on language modelling and data compression tasks.

Impact Statement

Predicting the next word in a sentence, or, more broadly, the next observation in a sequence, is a fundamental problem within statistics, computer science, and engineering with countless practical applications. Consider for example a mobile phone text entry system that assists the user by presenting options for possible continuations, or a speech recognition system that only needs to correctly understand a small part of a spoken utterance in order to complete the full sentence. The models described in this thesis are of this form: from a given data set of sequential observations they build a model of the statistical regularities in the data which can subsequently be used to make probabilistic predictions about the next symbol following some context of previous observations. Specifically, for text data and for data compression applications, the described models provide improvements over similar models in terms of accuracy and computational efficiency.

Outside of academia, the work presented in this thesis could, for example, be deployed to improve the performance of adaptive predictive text entry systems on mobile devices, or, more broadly, to improve assistive technology. Other commercial applications are e.g. constructing specialized compression algorithms for low-bandwidth settings, or using the models as a building block to address higher-level tasks (e.g. machine translation or speech recognition), by integration of the presented models and algorithms into the corresponding devices and applications. Within academia, the presented models could lead to the development of new models in the fields of machine learning, Bayesian nonparametrics, and natural language processing, could be used as building blocks to solve higher-level problems in these areas, or could be applied to address problems in adjacent domains (e.g. computational biology).

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Predicting what comes next in a sequence of observations is a fundamental problem in statistics and machine learning that comes in many forms and has countless applications. It is thus not surprising that a large number of models, algorithms, and heuristics to address various forms of this problem have been developed over the last century.

1.1 Problem Setting

In this thesis the focus is on a particular class of probabilistic *discrete time, discrete space* models, i.e. models that describe probability distributions over sequences $x_{1:N}$, where the individual observations x_i are associated with discrete positions $i = 1, \ldots, N$ and each x_i comes from a discrete (finite or countably infinite) set of possible symbols $x_i \in \Sigma$. One example of this setting and the primary application that will be considered in this thesis is *language modelling*, where the the *alphabet* Σ is the set of words in a language (e.g. English) and the sequences $x_{1:N}$ are sentences or documents. Assigning probabilities to sequences of words is a fundamental building block of many modern approaches to natural language processing (see Section 2.2). One special property of such natural language sequences (and also several other types of sequences) is that the empirical marginal and conditional distributions that are observed when examining large corpora of text tend to be well-approximated by power-law distributions. The models that are described and extended in this thesis incorporate this observation by utilizing a prior distribution in a hierarchical non-parametric Bayesian model that reflects this property.

Bayesian nonparametrics has received increased interest from the machine learning community over the last two decades as it provides an expressive

modelling framework that incorporates probability distributions over infinite-dimensional objects such as functions or probability measures as basic building blocks for model construction. This is especially well suited to the construction of *hierarchical* models, where these basic building blocks are arranged in hierarchies that express the modelers' beliefs about how information is shared between different parts of the hierarchy. The models described in this thesis will make use of such hierarchies to alleviate extreme data sparsity problems and will exploit the hierarchical structure to infer millions of (related) probability distributions, even if only a single observation is given for each of them.

The main advantage of the Bayesian approach over well-designed heuristics that are commonly used in the language modelling setting is that it (a) provides a principled way of incorporating prior information (such as the power-law nature of the conditional distributions) and (b) provides a principled way of handling uncertainty in model parameters when making predictions, namely by averaging with respect to the posterior distribution. Even though the ultimate goal of making predictions with a Bayesian model by marginalizing out all unknown quantities is often very difficult to attain for large and complex models, the Bayesian paradigm provides a framework for devising approximations, algorithms, and heuristics that trade off competing requirements such as storage limits, computational complexity, and parallelizability. By using the "optimal" solution as a guide, novel algorithms can be devised and existing algorithms can be analyzed and potentially improved if they can be described as approximations to Bayesian computations. This is the route taken in this thesis: several approximate inference algorithms with different strengths and weaknesses are developed and compared, and the resulting algorithms are shown to be scalable and effective.

1.2 Research Context & Related Work

The work in this thesis sits at the intersection of statistical language modelling, which is concerned with assigning probabilities to sequences of words, and the sub-field of machine learning based on Bayesian nonparametrics, which employs Bayesian models defined on infinite-dimensional parameter spaces to solve learning problems. Closely related models and techniques have also been developed in the field of lossless data compression, where probabilistic models are combined with entropy encoding techniques to yield effective lossless compression algorithms.

The foundation for work presented here lies in Yee Whye Teh's work on

using a hierarchical prior constructed from Pitman-Yor processes for language modelling [Teh, 2006a,b; Goldwater et al., 2006a], in which he not only showed that these models achieve state-of-the-art performance, but also pointed out a connection to the successful and widely-used Kneser-Ney smoothing approach to language modelling [Kneser and Ney, 1995], which was the de-facto standard at the time. Due to this connection, the work presented here is also closely related to classical language modelling techniques (see [Chen and Goodman, 1999] for an excellent review). In data compression, the PPM family of algorithms is in turn closely related to Kneser-Ney smoothing, and thus to the Pitman-Yor process-based language models. This connection has been pointed out in [Cowans, 2006] and explored further in detail in [Steinruecken, 2014]. Steinruecken [2014] also pointed out further connections of our HPYP-based compression algorithm (Chapter 6) to related compression techniques with unbounded-depth context trees and compared their performance. More recently, building on some of the work presented here, and also exploiting the connection between Kneser-Ney smoothing and the HPYP, Shareghi [2017] explored different ways of efficiently storing and performing inference in similar models.

In the field of machine learning there is large body of related work on Bayesian nonparametric models that have been shown to perform well in practice (see [Teh and Jordan, 2010] for an overview). These models make use of nonparametric Bayesian constructions such as Dirichlet processes [Ferguson, 1973; Teh, 2010], hierarchical Dirichlet processes [Teh et al., 2006] and their extension to Pitman-Yor processes, the Indian Buffet process [Griffiths and Ghahramani, 2011], or Gaussian processes [Rasmussen and Williams, 2006].

Another set of machine learning techniques that have been applied with tremendous success to both language modelling and data compression are artificial neural networks. While neural network approaches to language modelling are not a new idea [Miikkulainen and Dyer, 1991; Bengio et al., 2003], interest in them has recently been rekindled by very good predictive performance results [Mnih et al., 2009; Mikolov et al., 2011; Zaremba et al., 2014], tremendous improvements in hardware performance and generally growing interest in neural network models due to their success in other areas such as computer vision [Krizhevsky et al., 2012] and reinforcement learning [Mnih et al., 2013; Silver et al., 2018]. While the Sequence Memoizer language model presented in Chapter 3 achieved state-of-the-art predictive performance when it was first published, it has since been eclipsed by deep-learning-based approaches. As an example, the Cache Sequence Memoizer model discussed in

Chapter 7 achieves a perplexity of 113.8 on the Penn Tree Bank corpus (outperforming e.g. a Kneser-Ney 5-gram + cache model reported at 125.7 in [Mikolov, 2012]). In his extensive empirical evaluation of recurrent neural network (RNN) language models, Mikolov [2012] reported a perplexity of 124.7 for a single RNN model, 101.0 for an ensemble of RNN models, and 89.4 for an ensemble of RNN, n-gram, and cache models. Since then, "pure" deep learning models have been making great progress: Zaremba et al. [2014] report a perplexity of 78.4, Gal and Ghahramani [2016] report 73.4, Merity et al. [2016] report 70.9, and Zilly et al. [2017] report 66.0 for improved variants of RNN models (see e.g. [Melis et al., 2017] for more details). Neural network language models have seen even bigger improvements when larger training corpora are used (e.g. [Chelba et al., 2013]), and recent attention-based transformer architectures [Vaswani et al., 2017] not only perform exceedingly well as language models [Radford et al., 2018, 2019], but can also be applied directly to higher-level NLP tasks (e.g. machine translation, see [Popescu-Belis, 2019] for a recent review).

The "count-based", "non-neural" language modelling techniques presented in this thesis are mostly complementary to these approaches. The hybrid neural/HPYP models presented in Chapter 8 describe one way of combining a neural language model (albeit a simple one) with Bayesian nonparametric language models. While the Bayesian nonparametric models presented here certainly have applications on their own (e.g. resource-constrained settings or when little training data is available), we see combinations of the ideas underlying these models with the ideas underlying the neural models as a fruitful area for future investigation.

1.3 Overview and Contributions

The work presented in this thesis grew out of an initial collaboration with Frank Wood, Cédric Archambeau, Lancelot James, and Yee Whye Teh that resulted in the development of the *Sequence Memoizer* model [Wood et al., 2009, 2011], an extension of the Hierarchical Pitman-Yor Process language model [Teh, 2006a,b] to unbounded context lengths by making use of an efficient context tree data structure and the previously not well known coagulation-fragmentation properties of the Pitman-Yor process. The very promising initial results led me to pursue this model further, addressing some of the shortcomings of the initial work, including the space and time complexity of the proposed model construction and inference algorithm. In particular, it lead to the development of an effective online inference and model construction algo-

rithm presented in [Gasthaus et al., 2010], that makes it possible to construct the model in a streaming fashion for large data sets and allows it to be used in settings where predictions must be made on-line. We explored the applicability of this online technique to lossless data compression in [Gasthaus et al., 2010], where we demonstrated that the model yields state-of-the-art predictive performance in this setting. Further, in [Gasthaus and Teh, 2010] we addressed the space complexity issue by proposing a *compact representation* and associated inference algorithms and evaluating the time/space trade-offs of different representations. In [Gasthaus and Teh, 2010] we also extended the model to allow a larger hyperparameter range, which lead to improved performance; we also provided an elementary proof of the coagulation-fragmentation properties. For these already published contributions this thesis contains additional details and experiments.

The work presented in the final chapters goes beyond the Sequence Memoizer model: Chapter 7 presents a model addressing the phenomenon of "burstiness" (locally increased likelihood of words and phrases re-occurring) by integrating a cache-like model using the idea of "forgetting counts" [Bartlett et al., 2010]. The model presented in Chapter 8 is an initial attempt to bridge the gap between recently very successful neural language models and count-based models (like the Sequence Memoizer) by using the graphical Pitman-Yor process framework [Wood and Teh, 2009] to combine both types of models into a hybrid model that goes beyond simple averaging, thus exploiting the strengths of both approaches.

The remaining chapters of this thesis are organized in an attempt to present the development of the models in a logical fashion, grouping related concepts and developments together and proceeding from simpler to more complex models. This necessitates that novel contributions and review material are sometimes presented together, and that material is not presented chronologically in the order in which it was originally published.

1.4 Outline

The rest of this thesis is organized as follows:

Chapter 2 reviews necessary background material: power law sequences, classical n-gram language modelling techniques, the Pitman-Yor process, and its hierarchical extension the hierarchical Pitman-Yor process (HPYP), which is the foundation for the models presented in subsequent chapters. This Chapter is a review of prior art and no new results are presented.

Chapter 3 covers the *Sequence Memoizer* model originally published in [Wood et al., 2009, 2011].

Chapter 4 describes several representations of the Chinese restaurant franchise [Teh et al., 2004, 2005, 2006] for the HPYP language model and the Sequence Memoizer. It describes the "compact representation" we proposed in [Gasthaus and Teh, 2010], along with the associated Gibbs-sampling-based inference procedure.

Chapter 5 discusses online inference in the HPYP and Sequence Memoizer models. It describes and expands upon the model construction and inference procedure originally published in [Gasthaus et al., 2010]. A novel, deterministic inference procedure based on "fractional customers", an idea originally proposed by Blunsom and Cohn [2011] in a different setting, is also described.

Chapter 6 describes (DE-)PLUMP, an application of the Sequence Memoizer model to data compression, as originally proposed in [Gasthaus et al., 2010].

Chapter 7 introduces a "cache" language model based on the Sequence Memoizer for modelling non-stationarity and burstiness, making use of a time-varying variant of the PYP based on the idea of *forgetting*, first introduced in the HPYP setting by Bartlett et al. [2010].

Chapter 8 describes several variants of a novel hybrid language model that combines a neural language model (the log-bilinear model introduced by Mnih and Hinton [2007]), with a PYP-based hierarchical model similar to the ones discussed in the previous chapters. Training of the model is performed in an iterative fashion, interleaving sampling-based inference in the HPYP component with optimization of the LBL cost function using noise-contrastive estimation [Mnih and Teh, 2012].

Chapter 9 concludes this thesis with a summary and an outlook on future work.

CHAPTER 2

Background

This chapter reviews background material and introduces the concepts necessary for understanding the models developed in the later chapters. It consists of four main parts: An overview of the power-law properties present in natural language that we are attempting to model; a review of the language modelling problem and some of the classical techniques that have been developed for solving it; a review of the Pitman-Yor process and its properties; and a review of the hierarchical Pitman-Yor process and its application to the language modelling problem.

2.1 Power-law Sequences

The term *power-law sequences* in the title of this thesis refers to power-law properties present in the marginal and conditional distributions of the modelled sequences. In particular, the ranked probabilities in the marginal and conditional distributions follow a power-law, as does the number of distinct types that appear in each context as the observed sequence gets longer. Power laws are present in many natural and artificial phenomena such natural language (see below), city sizes, earth quake magnitudes, and book sales (see e.g. [Newman, 2005] for more examples) and estimating and modelling these properties has received significant attention. Explicitly incorporating these power-law properties into probabilistic models is an active area of research and has yielded improvements in the areas of graph modelling [Barabási and Albert, 1999], matrix completion [Meka et al., 2009], and language modelling [Goldwater et al., 2011; Teh, 2006b].

The term *power law* generally refers to a functional relationship of the form

$$f(x) = ax^{-g} \tag{2.1}$$

which has the main characteristic that the output scales proportionally when the input is scaled, i.e. $f(cx) = a(cx)^{-g} = c^{-g}ax^{-g} = c^{-g}f(x)$. As $\log f(x) = \log a - g \log x$ is a linear function of $\log x$ with slope $-g$, power law relationships are often plotted on log-log plots, where they appear as straight lines.

The term *power-law probability distribution* is used to refer to a distribution whose probability mass (or density) function (asymptotically) is a power law, i.e.

$$p(k) \propto k^{-g} \tag{2.2}$$

for $g \geq 1$. In the discrete case, the Zeta distribution, also referred to as Zipf's distribution, has exactly this form

$$p(k) = \frac{k^{-g}}{\zeta(g)} \qquad k = 1, 2, 3, \ldots \tag{2.3}$$

where the normalizing constant $\zeta(g) = \sum_{k=1}^{\infty} k^{-g}$ is the Riemann zeta function.

2.1.1 Power Laws in Natural Language

One widely known power-law property of natural language is known as *Zipf's law*: the ranked word frequencies in a sufficiently large corpus of text follow a power law, i.e. $c_{w(k)} \propto k^{-g}$, where $c_{w(k)}$ is the frequency of the k-th most common word and $g = 1$ in the original form of Zipf's law, but empirically typically found to be $1 \leq g \leq 2$ (see [Piantadosi, 2014] and references therein). A related power-law property is that the probability of observing a word with a particular frequency c_w is proportional to $c_w^{-g'}$. As noted e.g. in [Adamic and Huberman, 2002], these two power law properties are two sides of the same coin: any power-law distribution produces power law ranked frequencies and vice versa, where if the exponent on the power law is g', the exponent of the ranked frequency distribution is $g = 1/(g' - 1)$.

Plotting the frequency of types that occur exactly k times emphasizes rare words, whereas plotting the relative frequency of a word according to its rank emphasizes frequent words. Figure 2.1 illustrates these different ways of showing the power law present in the word frequencies of natural language (English in this case).

This power-law behavior can not only be observed in the marginal distribution of words, but is also present in the distribution of n-tuples (also called n-grams) of consecutive words, as well as the conditional distributions of words following a given *context* (a sequence of other words). Figure 2.2 shows the ranked (relative) frequency distributions of the marginal and several conditional distributions, and a similar power-law pattern can be observed in all

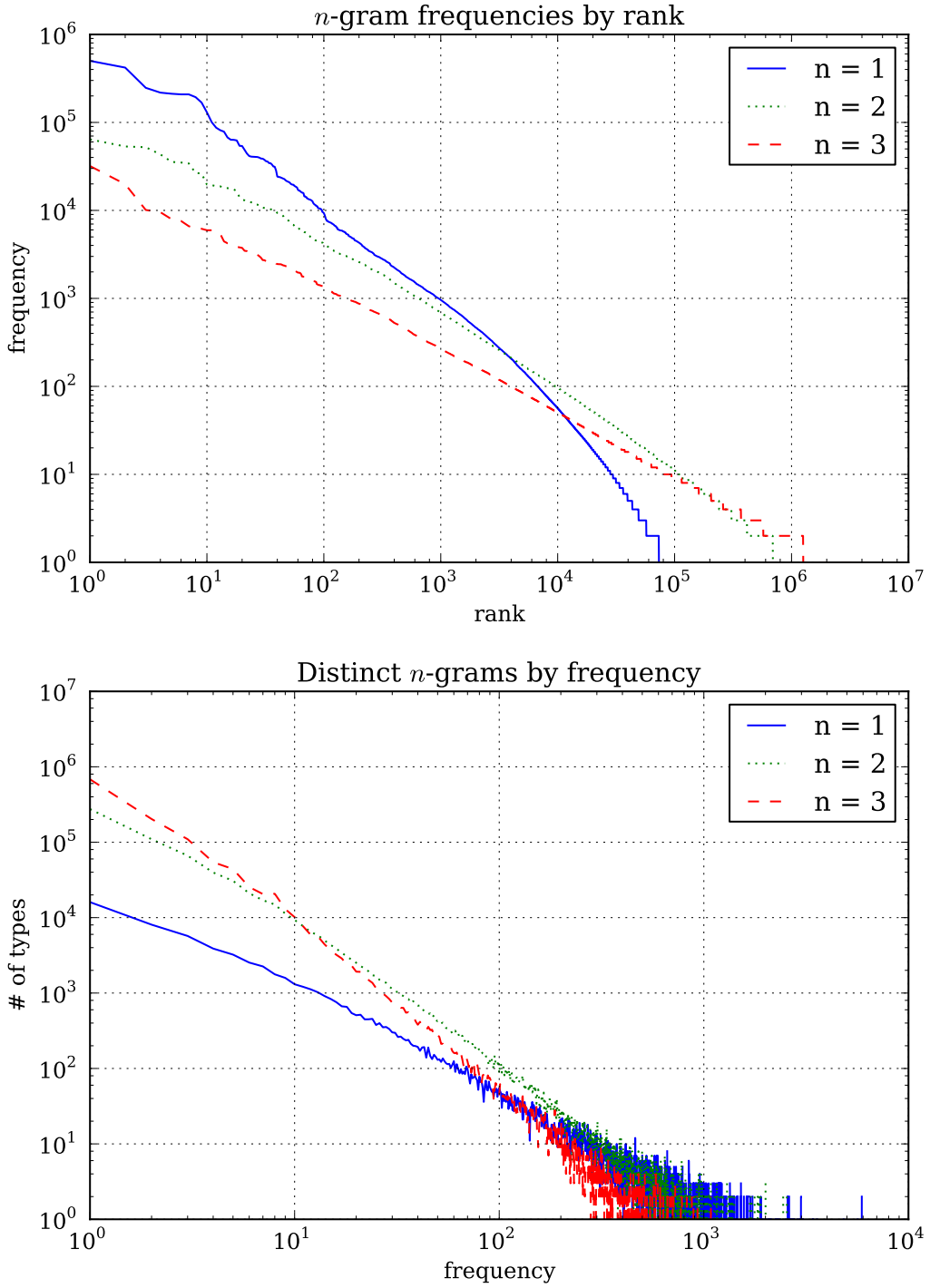


Figure 2.1: Power-law properties of the n-gram frequencies in a subset of the New York Times corpus (10 million tokens). The top panel shows the absolute n-gram frequencies (for $n = 1, 2, 3$) against rank (when sorted by frequency), while the bottom panel shows the number of distinct n-grams that occur with a given frequency. These two views of the n-gram frequencies emphasize different aspects: the plot according to rank emphasizes frequent n-grams, while the right plot highlights the existence of a large number of infrequent n-grams.

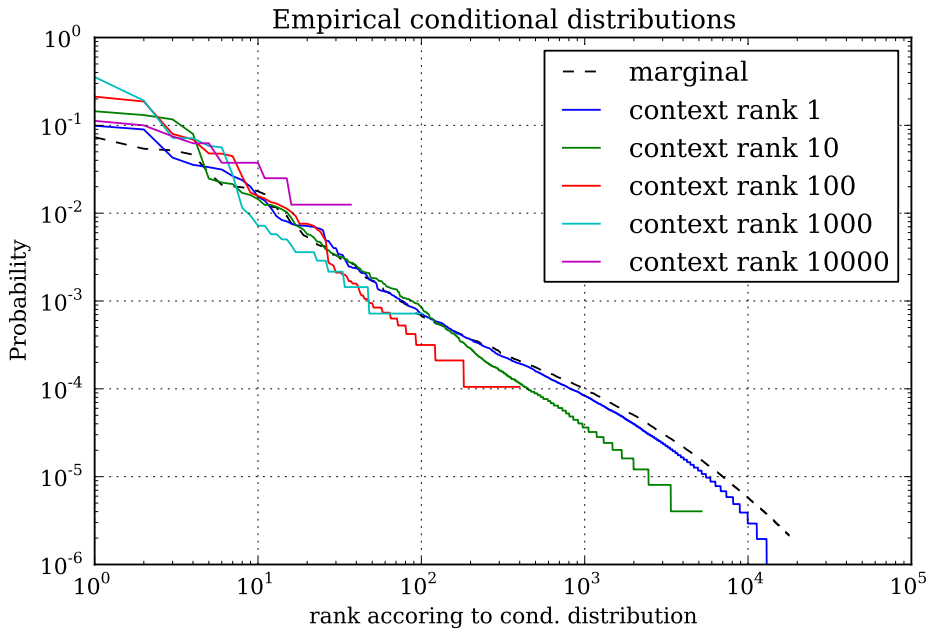


Figure 2.2: The empirical conditional distributions following the 10^i-th most frequent context word for $i = 0, 1, 2, 3, 4$ in the AP news corpus. The ordering of the target words on the x-axis is determined individually for each context word. For comparison, the dashed line shows the marginal distribution over words in the entire corpus.

of them. Further, the number of unique types that occur as one examines a sequence of more and more tokens also follows a power-law. This is shown in Figure 2.3, which plots the number of unique types against the number of tokens seen for three different corpora of English text. While we have demonstrated these power-law patterns using corpora of English text here, the same patterns can be observed for corpora of other languages, although in general the exponents of the power-laws will be different.

2.2 Language Modelling

The term *language model* is used to describe a method that assigns probabilities $\mathrm{P}(x_{1:N})$ to sequences of tokens (usually words), $x_{1:N}$.[1] Equivalently, such a model can compute the predictive probability distribution $\mathrm{P}(x_i \mid x_{1:i-1})$ for the next word x_i given a *context* of previous words $x_{1:i-1}$, as by the product law of probability we have

$$\mathrm{P}(x_{1:N}) = \prod_{i=1}^{N} \mathrm{P}(x_i | x_{1:i-1}). \tag{2.4}$$

The individual tokens x_i come from a fixed vocabulary of symbols Σ (also referred to as the *alphabet*) which is constructed ahead of time and usually

[1] In the most common language modelling setting the tokens are words, an we will use "token" and "word" interchangeably in the remainder.

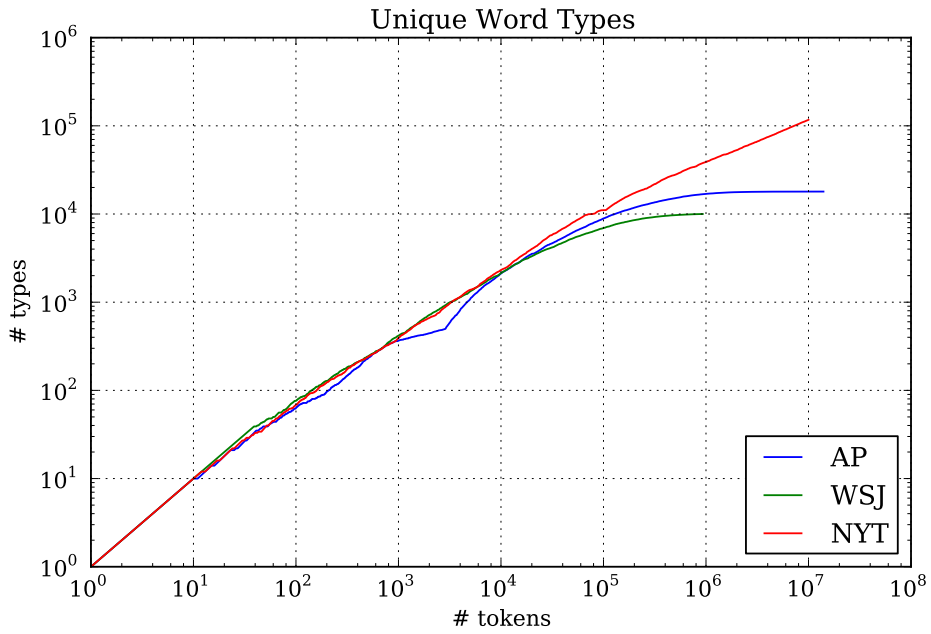


Figure 2.3: Number of unique words observed as a function of the number of word tokens observed for three different corpora. The levelling off of the growth for the AP and WSJ corpora is due to their pre-processing, which truncated the vocabulary to the most frequent ≈16K and 10K words respectively.

contains special tokens for symbols and numbers, as well as a special "out of vocabulary" token assigned to words that do not occur in Σ. A typical way to obtain Σ is to take all the tokens that occur more than k times in the training set, with $k \geq 1$.

Such probabilistic language models are the foundation (or a crucial component) of many very successful probabilistic approaches to machine translation [Brown et al., 1990], optical character recognition, handwriting recognition, speech recognition, spelling correction, predictive text entry, and information retrieval (see e.g. [Manning and Schütze, 1999] for an introduction).

Driven by initial success, language modelling has been an active area of research for many decades, with early research on the subject dating back to code breaking efforts during the second world war [Good, 1953]. Since then tremendous progress has been made, though so-called n-gram models (see Section 2.2.2), which were developed in the 1950s, are still at the heart of many modern techniques. On the one hand, the exponential increase in compute power and storage capacities combined with the availability of massive corpora of written language obtained from the internet [Brants and Franz, 2006; Chelba et al., 2013] have made building ever bigger models possible [Pauls and Klein, 2011; Heafield et al., 2013], yielding significant improvements in accuracy even with comparatively simple models. On the other hand, methodological

advances have made it possible to make better use of the available training data, so that accurate models can be built when less data or memory is available and have yielded novel ways for combining and adapting models to the specific problem setting. Many techniques from other areas of machine learning and statistics have been successfully applied to the language modelling problem, such as neural networks [Bengio et al., 2003; Mikolov, 2012; Mnih and Teh, 2012], random forests [Xu and Jelinek, 2004], and maximum entropy models [Rosenfeld, 1994]. See [Chen and Goodman, 1999] for an extensive review of n-gram language modelling techniques, [Goodman, 2001a] for a review of various extensions to these models such as clustering, mixture models, ensembles, and caching, and [Mikolov, 2012] for a recent extensive empirical evaluation of ensembles of various methods, including recent techniques using recurrent neural networks.

Interest in language modelling research mainly stems from the fact that improvements to language modelling techniques translate rather directly to improvements in natural language processing applications that use language models as a building block. Statistical machine translation is one such area where improvements in the language model's accuracy (usually measured in terms of *perplexity*, see below) typically result in improved translation accuracy (e.g. measured in terms of BLEU score).

Language modelling techniques have also been developed and applied in another area of computer science: lossless data compression. With the development of arithmetic coding [Pasco, 1976; Rissanen, 1976; Rissanen and Langdon, 1979; Witten et al., 1987] it has become possible (and practical) to convert any method for making one-symbol-ahead predictions $\mathrm{P}(x_i|x_{1:i-1})$ into a compression technique with only a negligible penalty incurred due to the coding. The performance of the resulting compressor thus relies on the predictive performance of the underlying probabilistic model. While the computational requirements for such models are somewhat different (fast online estimation is essential), the techniques that have been developed independently in this setting are similar to the techniques developed for natural language text, and sometimes even identical (as in the case of Kneser-Ney smoothing and the PPM-D compression algorithm, see below). This connection between lossless compression and language modelling has been explored by Cowans [2006] and Steinruecken [2014] (among others), an we will discuss it in more detail in Chapter 6.

2.2.1 Evaluating Language Models

The performance of a language model can be measured either in terms of its predictive accuracy independent of other components in an NLP system (*intrinsic evaluation*), or in terms of some performance measure for the overall system, such as word error rate of a speech recognizer (*extrinsic evaluation*). While the performance measure used in an extrinsic evaluation is problem-dependent, the de-facto standard for the intrinsic evaluation of language models is in terms of *perplexity* (or equivalently in terms of cross-entropy or log-loss, which is related via a monotonic transformation).

Logarithmic Loss & Perplexity

A standard technique for evaluating the predictive performance of a probabilistic model is to compute the probability the model assigns to an unseen test set. Taking the negative logarithm of this number and dividing by the length of the test sequence one arrives at the per-symbol logarithmic loss of a model P on a (test) sequence $x_{1:N}$, which is computed as

$$\ell = \frac{1}{N} \sum_{i=1}^{N} -\log_2 \mathrm{P}(x_i | x_{1:i-1}). \tag{2.5}$$

This quantity has an information-theoretic interpretation: it measures the average number of bits needed to encode each symbol in the sequence using P (modulo coding overhead). It is thus equivalent to the *bits per symbol* (bps) measure used in the data compression community if the sequence were encoded using an optimal encoder based on P.[2] A measure that is more commonly reported in the language modelling literature is *perplexity*, which is simply exponentiated average log-loss

$$\mathrm{PPL} = 2^{\ell} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{\mathrm{P}(x_i | x_{1:i-1})}}, \tag{2.6}$$

where the final expression gives another interpretation of perplexity as the geometric mean of the inverse probabilities. The main advantage of using perplexity over average log-loss is that it brings typical performance numbers into the more human-friendly range of tens and hundreds, though at the same time it distorts relative improvements and makes small improvements appear

[2] A *symbol* in the data compression context is usually a byte. The bps measure also typically includes all coding overhead and can be computed even for non-probabilistic coding schemes simply by diving the compressed size by the input size and multiplying by 8. Also note that in data compression there is usually no separate training set; the model is learned online as the sequence is processed sequentially.

bigger [Mikolov, 2012]. It also has the intuitive reference point that a uniform distribution over K symbols has a perplexity of K (instead of a log-loss of $\log_2 K$). Historically, perplexities have been quoted more frequently in the language modelling literature while log-loss is more common in other areas. Both measures will be used throughout this thesis, and one can easily convert between them by exponentiation or taking a base-2 logarithm.

In order to estimate the models predictive accuracy on unseen data (and not just its ability to memorize the training data), all performance measures have to be computed on a *test set* that hasn't been used in the training or tuning of the model. It is thus customary to split the available corpus data into a training, a validation, and a test set, and for some corpora traditionally used for comparing language models fixed standard splits have been established to ease comparison of results (see Appendix B for an overview of the datasets used here).

Evaluation of language models is typically performed offline, i.e. the model is not updated while making predictions on the test sequence and we mostly follow this tradition to keep our results comparable. For models which locally adapt to the context however, it makes sense to also perform an online evaluation, where the model can adapt as it predicts the test sequence symbol by symbol. We also follow previous work and evaluate the models on the entire test sequence (an entire corpus) and don't break it into smaller units such as paragraphs or sentences, as would perhaps be more realistic for some applications.

Caveats

While performing only an intrinsic evaluation of a language model in terms of perplexity does provide a measure of predictive accuracy that is comparable across studies as long as the exact same data sets are used, and is the de facto standard way to compare different models in the literature, there are some caveats that should be pointed out: Firstly, great care must be taken to not accidentally "cheat" in such an evaluation i.e. by using probabilities that are not properly normalized or by using information to make a prediction at step n that is only available in hindsight. A more practical concern is that in order for results to be comparable across studies, the data sets used must be exactly identical; two data sets based on the same corpus but pre-processed in different ways (e.g. by adding start/end-of-sentence tokens, converting some sequences to special tokens, or truncating the vocabulary) can yield vastly different perplexity results. The performance characteristics of different

algorithms can also change drastically with different training set sizes: a simple interpolated 5-gram model might perform very well when trained on a dataset of a trillion words, but perform poorly when only trained on a million words, while another model might not see a lot of improvement from adding more training data. Relative prediction performance of different algorithms can also vary with vocabulary size, which is problematic if vocabularies are artificially truncated for reasons of computational efficiency. Finally, an evaluation in terms of prediction performance does not assess other attributes that could be important in evaluating whether a model or algorithm is suitable for a particular task, such as scalability of training to large corpora, training speed (which can be on the order of weeks for some neural network models), prediction speed (especially important for settings where a large number of predictions are made in an "inner loop" of larger system, such as translation or speech recognition), memory requirements (e.g. for embedded applications like predictive text entry on mobile phones), and so on.

Despite these shortcomings, only performance figures in terms of perplexity on standard data sets (see Appendix B) will be reported in this thesis, and some of the other mentioned attributes will also be discussed. However, extrinsic evaluations in terms of application performance measures are beyond the scope of this thesis.

Why is Language Modelling Hard?

The fact that language modelling is still an active field of study even after several decades of research points to the fact that it is fundamentally a hard problem. Language has rich syntactic and semantic structure, but modelling and inferring this structure effectively and efficiently has proven to be a very difficult task. While there have been some successes in exploiting syntactic structure for language modelling [Chelba and Jelinek, 1998, 2000; Roark, 2001; Emami and Jelinek, 2005; Dyer et al., 2016], these models tend to be complex and not yet scalable to large corpora, limiting their applicability to small-scale settings. Similarly, class-based language models which try to explicitly capture semantic relationships between individual words have not yielded the hoped-for improvements over simpler models.

Language models based on sequence models that do not attempt to explicitly model any further hierarchical syntactic structure or structure in the vocabulary, such as n-gram models or the models described in this thesis, face the difficulty of learning from data sets that are small compared to size of the space of possible sequences, which grows exponentially in the sequence

length. For example, even when restricting to a 10,000 word vocabulary, there are $(10^4)^5 = 10^{20}$ possible sequences of length 5, while the largest available language corpora (e.g. the Web 1T 5-gram corpus [Brants and Franz, 2006]) contain "only" 1 trillion (10^{12}) tokens. While of course many of the possible sequences will never appear because they are not grammatical, the fundamental difficulty of language modelling lies in accurately estimating the probability of unseen sequences, while at the same time effectively exploiting the relative frequency information for sequences that were observed before. Because of this, many classical language modelling techniques center around estimating how much probability mass to "reserve" for tokens that have not been observed in a given context, and how to distribute this reserved mass among the remaining tokens. Witten and Bell [1991] refer to this fundamental difficulty as the "zero-frequency problem".

2.2.2 n-gram Models

An n-gram model, also known as a Markov model of order $n-1$, is a collection of conditional distributions over words $\{\mathrm{P}_{\mathbf{u}}(x)\}_{\mathbf{u}\in\Sigma^{n-1}}$, one for each possible context $\mathbf{u} \in \Sigma^{n-1}$ of $n-1$ words. The probability of a sequence $x_{1:N}$ under such a model is given by

$$\mathrm{P}(x_{1:N}) = \prod_{i=1}^{N} \mathrm{P}_{\sigma_{n-1}(x_{1:i-1})}(x_i) \tag{2.7}$$

where $\sigma_k(x_{1:N}) = x_{N-k+1:N}$ denotes the length-k suffix of $x_{1:N}$.[3] The core idea underlying these models is that any context that appeared more than $n-1$ words before the word that is being predicted does not influence the prediction, so that $\mathrm{P}(x_i|x_{1:i-1}) \approx \mathrm{P}(x_i|\sigma_{n-1}(x_{1:i-1}))$. Such a model can be estimated from n-tuples (or *n-grams*) or words, giving them their name.

A straightforward way to estimate these conditional distributions is to set them equal to the empirical conditional distributions on the training set, which can equivalently be phrased as the maximum likelihood estimate of the parameters of a categorical distribution, and amounts to simple ratios of counts (see e.g. [Manning and Schütze, 1999]):

$$\mathrm{P}^{\mathrm{MLE}}_{\mathbf{u}}(x) = \frac{\#(\mathbf{u}x)}{\#(\mathbf{u})} \tag{2.8}$$

where $\#(\mathbf{u})$ counts the number of times $\mathbf{u}$ appears in the training set.

Given an infinite training corpus and infinite computational resources one could accurately estimate such a model for some large n and obtain an "op-

[3] The boundary case at the beginning of the sequence can be dealt with by either left-padding the sequence with a special token, or by separately estimating sets of distributions for all context lengths up to $n-1$.

timal" model.[4] However, in practice the simple estimator in (2.8) has several problems related to data sparsity: As one increases n in order to capture dependencies between words further apart, the number of possible contexts grows exponentially and one quickly (already for $n = 3$) encounters the situation that $\#(\mathbf{u}x)$ or $\#(\mathbf{u})$ is zero for some $\mathbf{u}$, i.e. the context or context-symbol pair does not occur in the training set. In the AP news corpus data set (see Section B.3) for example, more than half of the unique trigrams in the test set do not occur in the training set. Having $\#(\mathbf{u}x) = 0$ yields $\mathrm{P}_{\mathbf{u}}^{\mathrm{MLE}}(x) = 0$ and thus $\mathrm{P}(x_{1:N}) = 0$ for any sequence that contains an unseen n-gram, which is at best undesirable for most applications. Even more problematic is the case $\#(\mathbf{u}) = 0$, for which the estimator is undefined. Addressing this "zero-frequency problem" [Witten and Bell, 1991] of having to assign a non-zero probability to a context-symbol pair that has not occurred in the training data is crucial for making language modelling useful. Even if $\#(\mathbf{u})$ is non-zero and all symbols x were observed at least once in the context, the resulting estimates might be very inaccurate if the symbols were only observed infrequently.

Most of the research on n-gram models has thus been focussed on overcoming these data sparsity problems by developing estimators that address the zero-frequency problem and are more robust when little training data is available in a given context. We will briefly review some of the classical techniques in the next sections, but refer the reader to [Chen and Goodman, 1999] for a detailed discussion and evaluation of many of these techniques. The hierarchical non-parametric Bayesian models presented later can be viewed as elaborate and well-motivated methods for combining these classic techniques.

Smoothing

A conceptually simple way to alleviate the "zero problem" for each conditional distribution independently is by allocating some probability mass to the symbols with zero count, thus "smoothing" the distribution. There are many ways to do this, and a large amount of research has been devoted to devising and evaluating different techniques [Chen and Goodman, 1999].

One of the simplest smoothing methods is known as *additive smoothing*. The idea is to simply add some constant α to each count $\#(\mathbf{u}x)$ (and re-normalizing appropriately),

$$\mathrm{P}_{\mathbf{u}}^{\mathrm{ADD}}(x) = \frac{\#(\mathbf{u}x) + \alpha}{\#(\mathbf{u}) + \alpha|\Sigma|} \tag{2.9}$$

[4] In fact, the fact that corpora and computers have been getting bigger exponentially fast has surely contributed to n-gram models remaining popular.

where $|\Sigma|$ is the size of the alphabet. This method can also be given a Bayesian interpretation, as (2.9) is equal to the expected value of the probability of x under the posterior distribution of a categorical-Dirichlet model with symmetric Dirichlet prior with parameter α [MacKay and Bauman Peto, 1995]:

$$x_i \overset{\text{iid.}}{\sim} \text{Disc}(\mathbf{p}) \tag{2.10}$$

$$\mathbf{p} \sim \text{Dir}(\alpha). \tag{2.11}$$

Intuitively, the larger α the more data is needed to overcome the prior (which is uniform over all symbols). Note that the effect of this smoothing method diminishes as more data is observed. This is desirable, but is not the case for all smoothing methods: for example, interpolating (2.8) with a uniform distribution over Σ also alleviates the zero problem, but introduces a bias that does not diminish with more data.

Mixtures & Interpolation

Another powerful technique for improving the performance of language models (and probabilistic models in general) is to combine multiple models into a *mixture* model. Given a set of base models $P_i(\cdot), i \in \mathscr{I}$ (for some index set $\mathscr{I}$) and non-negative weights λ_i, $\sum_{i\in\mathscr{I}} \lambda_i = 1$, we can define a new model $\mathrm{P}^{\text{mix}}(\cdot)$ as a convex combination of the base models:

$$\mathrm{P}^{\text{mix}}(\cdot) = \sum_{i\in\mathscr{I}} \lambda_i \mathrm{P}_i(\cdot). \tag{2.12}$$

This simple idea is fundamental to many advanced language modelling techniques, though they are sometimes not explicitly cast in this form.

A natural starting point for applying this technique to n-gram language models is to create a mixture (or in other words *interpolate*) between the conditional distributions of an n-gram model and an $(n-1)$-gram model, i.e.

$$\mathrm{P}^{\text{interp}}_{\mathbf{u}}(x) = \lambda_{\mathbf{u}} \mathrm{P}_{\mathbf{u}}(x) + (1-\lambda_{\mathbf{u}}) \mathrm{P}_{\sigma(\mathbf{u})}(x) \qquad \forall \mathbf{u} \in \Sigma^{n-1} \tag{2.13}$$

where we denote by $\sigma(s\mathbf{u}) = \mathbf{u}$ the *longest proper suffix* of a sequence (i.e. applying $\sigma(\cdot)$ removes the first symbol). The mixture weight $\lambda_{\mathbf{u}} \in [0,1]$ can depend on the context $\mathbf{u}$, but in practice these weights are usually tied together in some way to reduce the total number of parameters that have to be estimated. The component models $\mathrm{P}_{\mathbf{u}}$ and $\mathrm{P}_{\sigma(\mathbf{u})}$ can be estimated by any technique, e.g. by employing the MLE (2.8) or additive smoothing (2.9).

This process can be repeated recursively, so that the lower-order model in such a mixture can itself be constructed as an interpolation with the next

lower order model. The union of all conditional distributions involved in such a construction $\{\mathrm{P}_{\mathbf{u}}(x)\}_{\mathbf{u}\in\Sigma^{<n}}$ can be arranged in a tree, where each node corresponds to a context $\mathbf{u}$, and the parent of each node is given by its longest proper suffix $\sigma(\mathbf{u})$, terminating with the empty context ε at the root of the tree (see Figure 3.1 for an illustration). Such context trees also underlie the hierarchical Bayesian models discussed later, and we will come back to them in Section 3.2.

If the individual conditional distributions in such a hierarchy are estimated independently using the maximum likelihood estimator (2.8), the resulting technique is known as Jelinek-Mercer smoothing [Jelinek and Mercer, 1980]. Many techniques for improving n-gram models are variants of this basic technique and differ in how the base models are estimated (e.g. by applying additional smoothing), and how the mixture weights $\lambda_{\mathbf{u}}$ are chosen. Mixture models can not only be used for blending higher- and lower-order models, but also to combine different classes of models, e.g. neural language models with n-gram models, in order to hedge against each individual model's weaknesses (while exploiting each model's strengths). We will discuss this idea in more detail Chapter 8.

Back-Off

An alternative method for combining a higher order n-gram model with a lower order one is known as *back-off*.[5] Back-off differs from interpolation (2.13) in that instead of affecting the distribution for all symbols, in a back-off combination the lower-order model is only used if $\#(\mathbf{u}s) = 0$:

$$\mathrm{P}^{\mathrm{BO}}_{\mathbf{u}} = \begin{cases} \mathrm{P}_{\mathbf{u}}(x) & \text{if } \#(\mathbf{u}s) > 0 \\ \mathrm{P}_{\mathbf{u}}(\star)\mathrm{P}_{\sigma(\mathbf{u})}(x) & \text{if } \#(\mathbf{u}s) = 0. \end{cases} \tag{2.14}$$

In order for such a model to be properly normalized, some probability mass of $\mathrm{P}_{\mathbf{u}}(x)$ has to be reserved for the $\#(\mathbf{u}s) = 0$ case, so that $P_{\mathbf{u}}(s)$ can't be estimated by (2.8) directly. One way of formalizing this requirement is to augment the alphabet with a special *escape symbol* $\star$ as we have done above. This symbol absorbs the extra mass, so that $\sum_{s\in\Sigma}\mathrm{P}_{\mathbf{u}}(s) + \mathrm{P}_{\mathbf{u}}(\star) = 1$. Note that further $\mathrm{P}_{\sigma(\mathbf{u})}(x)$ needs to satisfy $\sum_{s\in\Sigma}\mathrm{I}[\#(\mathbf{u}s) = 0]\cdot\mathrm{P}_{\sigma(\mathbf{u})}(x) = 1$, i.e. it needs to be normalized over the set of symbols for which the higher-order model does not make predictions. The various back-off techniques that have been proposed differ in how much

[5] The term *back-off* is also sometimes used in a less strict sense referring to some way of combining a complex model with a simpler one, even if the mechanism used for combining the models is different. In the data compression literature the same technique is sometimes referred to as using an *escape symbol*.

mass they assign to $\star$, and how the MLE (2.8) is modified to make $\sum_{s\in\Sigma} \mathrm{P}_{\mathbf{u}}(s) < 1$. One approach (employed in the PPM-A [Cleary and Witten, 1984] compression algorithm) is to rescale the MLE and set $\mathrm{P}_{\mathbf{u}}(x) = (1 - \mathrm{P}_{\mathbf{u}}(\star))\,\mathrm{P}^{\mathrm{MLE}}_{\mathbf{u}}(x)$ where $\mathrm{P}_{\mathbf{u}}(\star)$ is set to $\mathrm{P}_{\mathbf{u}}(\star) = 1/\left(1 + \sum_s \#(\mathbf{u}s)\right)$ (see [Cowans, 2006, Chapter 2]). Back-off can be preferable over interpolation for computational reasons, especially in settings where models must be constructed and scored on-line (such as data compression): In (2.14) the lower-order model only has to be evaluated if $\#(\mathbf{u}s) = 0$, whereas in (2.13) it has to computed for every symbol. If the recursively constructed back-off or interpolation hierarchies are deep, this can lead to substantial computational savings. However, their predictive performance is often worse than their interpolated counterparts (as discovered for Kneser-Ney by Chen and Goodman [1999] and for PPM-D by Steinruecken [2014]), and they are more difficult to analyze and implement as the distributions involved need to be re-normalized appropriately.

2.2.3 Absolute Discounting & Kneser-Ney Smoothing

One particularly effective way of estimating interpolated n-gram models is known as *Kneser-Ney* (KN) smoothing [Kneser and Ney, 1995].[6] Kneser-Ney smoothing is a variant of a technique known as *absolutely discounting* [Ney et al., 1994], and is of particular relevance to the models discussed later in this thesis as it can be given a Bayesian interpretation in terms of a hierarchical Pitman-Yor process model [Teh, 2006a,b]. The idea of *discounting* is that in order to "reserve" some probability mass for previously unseen symbols (i.e. the mass assigned to $\star$ in the escape symbol formulation), one has to take away (discount) some probability mass from the observed symbols (relative to the maximum likelihood estimate (2.8)). The mixture distribution (2.13) or the PPM-A technique described above can be seen as a form of *relative* discounting, where the probability of each symbol under the higher-order model is decreased by a constant factor, thus making the amount of probability mass taken away from each symbol proportional to its probability. The idea of *absolute* discounting on the other hand is to deduct a *constant amount* of probability mass that is independent of the symbol's probability. This reserved probability mass is then re-distributed according to a lower-order model. The

[6] Kneser and Ney [1995] originally introduced it as a back-off procedure, but Chen and Goodman [1999] describe an interpolation variant of this technique that is easier to derive and has better empirical performance. When we refer to Kneser-Ney smoothing in this thesis we are referring to this variant (which Teh [2006a] refers to as "Interpolated Kneser-Ney").

resulting estimator has the form

$$\mathrm{P}_{\mathbf{u}}^{\mathrm{disc}}(x) = \frac{\max(\#(\mathbf{u}x) - d, 0)}{\#(\mathbf{u})} + \frac{d\tilde{t}_{\mathbf{u}\cdot}}{\#(\mathbf{u})} P_{\sigma(\mathbf{u})}^{\mathrm{base}}(x) \tag{2.15}$$

where the $d \in [0,1]$ is the *discount parameter*. We have further defined $\tilde{t}_{\mathbf{u}s} = I[\#(\mathbf{u}s) > 0]$ and $\tilde{t}_{\mathbf{u}\cdot} = \sum_{s\in\Sigma} \tilde{t}_{\mathbf{u}s} = \sum_{s\in\Sigma} I[\#(\mathbf{u}s) > 0]$, i.e. $\tilde{t}_{\mathbf{u}\cdot}$ counts the number of symbols appearing after context $\mathbf{u}$ in the training set. This factor ensures that the distribution is properly normalized, as the discount d is subtracted from the left term for each of the $\tilde{t}_{\mathbf{u}\cdot}$ symbols with $\#(\mathbf{u}s) > 0$.[7] The distribution P^{base} is the *base distribution* that is used for re-distributing the discounted probability mass.

Kneser and Ney [1995] proposed that P^{base} in this setting should be estimated such that the marginal probabilities under the resulting model match the marginal statistics of the training corpus, and they showed that this can be achieved by estimating the lower-order model from modified counts. Arriving at the same result as Kneser and Ney [1995], Chen and Goodman [1999] showed that in order to achieve this consistency with the empirical marginal distribution for a bigram model estimated according to (2.15), $P_{\sigma(\mathbf{u})}^{\mathrm{base}}(x)$ should be estimated as

$$P_{\mathbf{u}}^{\mathrm{KN}}(x) = \frac{\tilde{t}_{\cdot\mathbf{u}x}}{\tilde{t}_{\cdot\mathbf{u}\cdot}} \tag{2.16}$$

where $\tilde{t}_{\cdot\mathbf{u}x} = \sum_s \tilde{t}_{s\mathbf{u}x}$ counts the number of contexts $s\mathbf{u}$ with suffix $\mathbf{u}$ in which x occurs, and $\tilde{t}_{\cdot\mathbf{u}\cdot} = \sum_x \tilde{t}_{\cdot\mathbf{u}x}$. The estimate (2.16) can be interpreted as the MLE (2.8) applied to modified counts, where $\#(\mathbf{u}s)$ is replaced with $\tilde{t}_{\cdot\mathbf{u}x}$. For higher-order n-gram models Kneser and Ney [1995] proposed applying (2.15) recursively to these modified counts. In other words, the "observations" from which the lower order distributions $P_{\mathbf{u}}(x)$ are estimated are not the number of times a given symbol x was observed in the context $\mathbf{u}$, but instead the number of unique symbols that precede the string $\mathbf{u}x$ in the corpus. This form of modified counts is also referred to "update exclusion" or "shallow updates" in the compression literature and in addition to improved predictive performance has a computational advantage in the online setting, as after observing the context-symbol pair $\mathbf{u}s$ only a subset of the associated counts has to be updated (see Chapter 6 and [Steinruecken, 2014, Chapter 6] for more details).

Ney et al. [1994] originally proposed setting the discount parameter to $d = n_1/(n_1 + 2n_2)$, where n_1 and n_2 are the number of n-grams that occur exactly ones or twice respectively. Chen and Goodman [1999] showed that by

[7] The notation $\tilde{t}_{\mathbf{u}\cdot}$ used here alludes to the Bayesian interpretation described later, where $t_{\mathbf{u}\cdot} = \sum_s t_{\mathbf{u}s} \geq \tilde{t}_{\mathbf{u}\cdot}$ is a random variable denoting the number of **t**ables in a Chinese restaurant process.

making the discount parameter dependent on the count $\#(\mathbf{u}s)$ it is subtracted from, accuracy could be improved. They proposed using three different discounts parameters for counts one, two, and three or above (different for each n), and showed that this algorithm (which they called "modified Kneser-Ney smoothing") with discount parameters optimized on a held-out set consistently outperformed all other methods considered in their comparison. The prediction model used in the PPM-D compression algorithm is virtually identical to the back-off version of Kneser-Ney smoothing, except that PPM-D uses a fixed discount $d = 1/2$, that is the same for all levels in the back-off hierarchy, i.e. independent of n. Steinruecken [2014] discovered that using level-dependent discount parameters is a crucial ingredient in making such models not degrade in performance as n gets larger, and points to this as the main reason why our variant of unbounded context length Kneser-Ney smoothing (UKN, see Section 5.2) outperforms other attempts to incorporate unbounded context into PPM.

Parameter Estimation

All of the techniques described above have free parameters (e.g. the additive smoothing constant α in (2.9), the mixture weights $\lambda_{\mathbf{u}}$ in (2.13), or the discount parameters d in (2.15)), that need to be set. For some of the classical language modelling techniques heuristics for setting these parameters have been devised (e.g. $\alpha = 1$ or $d = n_1/(n_1 + 2n_2)$ for the discount parameter in [Ney et al., 1994]). However, when computationally feasible, the predictive performance can typically be improved by optimizing these parameters based on data, e.g. by minimizing the negative log-likelihood on a hold out set with respect to these parameters (which we will collectively refer to as θ):

$$\hat{\theta} = \arg\min_{\theta} -\sum_i \log \mathrm{P}_\theta(x_i|x_{1:i-1}) \tag{2.17}$$

Note that for the techniques presented above, this optimization must be performed on a validation data set that is disjoint from the training data set used for obtaining the counts. If one were to optimize α in (2.9), $\lambda_{\mathbf{u}}$ in (2.13), or d in (2.15) on the training set, one would obtain $\alpha = 0$, $\lambda_{\mathbf{u}} = 1$, and $d = 1$ respectively as the optimal value, i.e. one would revert to the maximum likelihood estimate (2.8).

An effective way to perform the optimization in (2.17) is to compute the gradient of the objective function with respect to θ,

$$\frac{\partial}{\partial\theta} -\sum_i \log \mathrm{P}_\theta(x_i|x_{1:i-1}) = -\sum_i \frac{1}{\mathrm{P}_\theta(x_i|x_{1:i-1})} \frac{\partial}{\partial\theta} \mathrm{P}_\theta(x_i|x_{1:i-1}) \tag{2.18}$$

and to use a gradient-based optimization procedure such as (stochastic) gradient descent, Newton's method, or a quasi-Newton method such as L-BFGS (see e.g. [Nocedal and Wright, 2006]). For constraint parameter values (e.g. $\alpha \geq 0$ and $0 \leq d \leq 1$) it is convenient to perform the optimization with respect to an unconstrained re-parametrization, e.g. by letting $\alpha = \log(1 + \exp(\tilde{\alpha}))$ or $d = 1/(1 + \exp(-\tilde{d}))$ and then performing an unconstrained optimization wrt. $\tilde{\alpha}$ or $\tilde{d}$ respectively.

One special case are the mixing weights λ_i of a mixture model which can alternatively be optimized effectively using the well-known EM algorithm [Dempster et al., 1977]. In the special case when the parameters of the base models are fixed this reduces to the following simple updates:

$$r_{ni} \leftarrow \lambda_i P_i(x_n|x_{1:n-1}) \qquad i \in \mathscr{I}, n = 1,\ldots,N \tag{2.19a}$$

$$r_{ni} \leftarrow \frac{r_{ni}}{\sum_{i' \in \mathscr{I}} r_{ni'}} \qquad i \in \mathscr{I}, n = 1,\ldots,N \tag{2.19b}$$

$$\lambda_i \leftarrow \frac{1}{N} \sum_{n'=1}^{N} r_{n'i} \qquad i \in \mathscr{I} \tag{2.19c}$$

where the first two steps correspond to the E-step (computing and normalizing the posterior distribution of the latent variables to obtain the so-called *responsibilities* r_{ni}) and the third step constitutes the M-step where the only parameters to maximize over are the mixture weights λ_i. These steps are repeated until convergence.

2.3 The Pitman-Yor Process

This section reviews the Pitman-Yor process (PYP) and its closely related cousins: the two-parameter Poisson-Dirichlet (PD) distribution and the two-parameter Chinese Restaurant Process (CRP). Results that are necessary for the developments in subsequent chapters are compiled from various sources (most notably the work of Pitman [Pitman, 1992, 1995, 1996; Pitman and Yor, 1997; Pitman, 2002]) and stated using the notation used in this thesis, but no new results are presented. Overview articles highlighting different aspects are available in [Carlton, 1999; Ishwaran and James, 2001; Buntine and Hutter, 2012].

We will discuss the basic properties in Sections 2.3.1–2.3.3, the power-law properties of the PYP in Section 2.3.4, and discuss Monte Carlo inference in basic PYP models, including the CRP-based add/remove Gibbs sampler that forms the basis of many MCMC schemes for PYP-based models in Section 2.3.6.

2.3.1 Definition

The Pitman-Yor process, denoted $\mathrm{PY}(\alpha, d, H)$, can be seen as a generalization of the more widely known Dirichlet process (DP) $\mathrm{DP}(\alpha, H)$ [Ferguson, 1973; Teh, 2010]: Both are measures over probability measures centered around a *base measure* H, and the DP arises as a special case from $\mathrm{PY}(\alpha, d, H)$ when $d = 0$. What is now commonly referred to as the *Pitman-Yor process* (PYP) was originally introduced in [Pitman and Yor, 1997] under the name *two-parameter Poisson-Dirichlet distribution* as an extension of the Poisson-Dirichlet distribution of Kingman [1975].[8]

A draw G from a Pitman-Yor process $G \sim \mathrm{PY}(\alpha, d, H)$ is a random discrete probability measure, most directly characterized by the following *stick-breaking construction* [Pitman, 1996; Ishwaran and James, 2001]:

$$G(\cdot) = \sum_{i=1}^{\infty} p_i \delta_{\theta_i}(\cdot) \tag{2.20}$$

[8] The name *Pitman-Yor process* was coined by Ishwaran and James [2001] to refer to the distribution induced by a size-biased permutation of the weights of the two-parameter Poisson-Dirichlet distribution introduced by Pitman and Yor [1997], i.e. the distribution given by the stick-breaking construction, sometimes also referred to as the (two-parameter) GEM (Griffiths, Engen, McCloskey) distribution. Here we will adopt common convention and call the distribution over random measures (2.20) the Pitman-Yor Process, the distribution over the weights $p_1, p_2, \ldots$ arising from the stick-breaking construction the GEM distribution, and the associated distribution over random partitions the (two-parameter) Chinese Restaurant Process (CRP). We reserve the name "two-parameter Poisson-Dirichlet distribution" for the distribution of the ranked probabilities $p_{(1)} \geq p_{(2)}, \geq \ldots$ as originally defined in [Pitman and Yor, 1997].

where

$$p_i = V_i \prod_{k=1}^{i-1} (1 - V_k) \tag{2.21a}$$

$$V_i \sim \text{Beta}(1 - d, \alpha + d\,i) \tag{2.21b}$$

and $\theta_i \sim H$. The parameter $\alpha \in (-d, \infty)$ is called the *concentration parameter*, $d \in [0, 1)$ the *discount parameter*, and H the *base measure*.[9] If the base measure H is a probability measure over some probability space $(\Sigma, \mathscr{F})$, then $G \sim \text{PY}(\alpha, d, H)$ is a *random* measure over the same space. Furthermore, for all $A \in \mathscr{F}$, $E[G(A)] = H(A)$, i.e. the base measure H can be thought of as the mean of G. This property makes it easy to construct hierarchical models using the PYP by making the base measure H itself a draw from a PYP.[10]

Equation (2.21) is called the stick-breaking construction and is denoted $\mathbf{p} \sim \text{GEM}(\alpha, d)$. The weights p_i in the sick-breaking construction are not in decreasing order. When they are sorted in decreasing order, the sorted weights $p_{(1)} \geq p_{(2)} \geq p_{(3)} \geq \ldots \sim \text{PD}(\alpha, d)$ follow a two-parameter Poisson-Dirichlet distribution [Pitman and Yor, 1997, Proposition 2]. Conversely, a *size-biased sample* from these sorted weights, where the weights are chosen sequentially proportional to their size and then removed from the possible options, is distributed according to (2.21) [Pitman and Yor, 1997, Proposition 2].

The Effect of α and d

It is instructive to gain some intuition for which aspects of the PYP are controlled by the parameters α and d. As already noted, for any $A \in \mathscr{F}$ we have $E[G(A)] = H(A)$, i.e. the mean of the random measure G is given by the base measure H. The variance of $G(A)$ for $G \sim \text{PY}(\alpha, d, H)$ is given by [Buntine and Hutter, 2012, Lemma 35]:

$$\text{Var}[G(A)] = \frac{1 - d}{\alpha + 1} G(A)(1 - G(A)) \tag{2.22}$$

so that—considering α and d separately—the variance grows as $1/\alpha$ and $1 - d$ respectively. This behavior is illustrated in Figure 2.4. Further, we will see in Section 2.3.4 how d controls some of the power law properties of the PYP.

[9] Unfortunately, the conventions used for naming these parameters differ between the machine learning literature and the probability theory and statistics literature, where the concentration parameter is commonly called θ and the discount parameter α (e.g. in the work by Pitman). The name "discount parameter" for d was coined by Teh [2006a,b] and stems from its connection to Kneser-Ney smoothing (Section 2.2.3), which will be discussed in Section 2.4.2.

[10] In all the models discussed in this thesis Σ will be a finite set. However, in general they can also be applied if Σ is countably infinite as long as a probability measure H can be defined and evaluated for each $s \in \Sigma$.

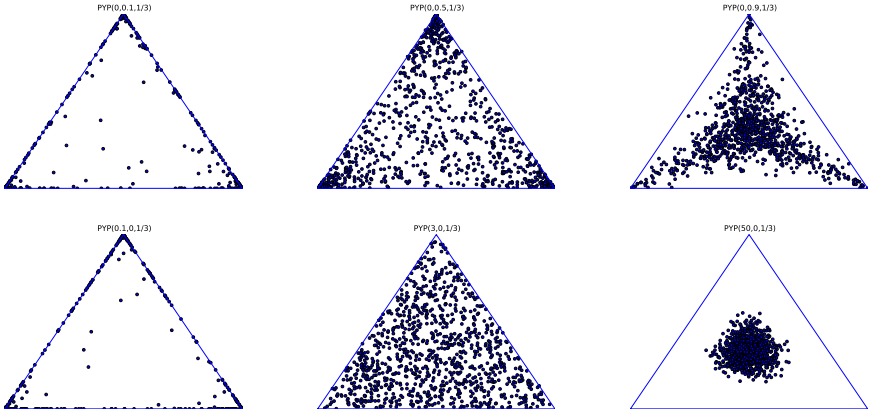


Figure 2.4: Samples from PY(α, d, Disc($\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$)) for varying values of α and d. Each black dot corresponds to one sample, which is a probability distribution over three outcomes, i.e. a vector (p_1, p_2, p_3) with non-negative entries satisfying $\sum_{i=1}^{3} p_i = 1$, which we plot on the unit triangle using barycentric coordinates. Top row: $\alpha = 0$ and $d = 0.1$, $d = 0.5$, $d = 0.9$ (left to right). Bottom row: $d = 0$ and $\alpha = 0.1$, $\alpha = 1$, $\alpha = 50$.

2.3.2 Two Parameter Chinese Restaurant Process

The (two-parameter) Chinese Restaurant Process (CRP) is a sequential construction for a distribution over the set of partitions the integers $[n] = \{1, \ldots, n\}$ that describes the partition structure that arises when sampling $X_j \sim G, j = 1, \ldots, n$ where G is PYP-distributed and hence has the form (2.20).[11] The possible values of X_j are the values θ_i drawn from the base distribution H in (2.20), and multiple X_j can correspond to the same θ_i.[12] The distribution over the set of partitions of the integers $[n]$ induced by grouping the X_j according to the draws θ_i from the base distribution H is precisely the one characterized by the two-parameter CRP [Pitman, 1995].

Before describing the CRP let us first introduce some notation: A *partition* $A \in \mathscr{A}_S^K$ of a set S is a decomposition of S into K non-empty disjoint subsets a_k (called *blocks*) such that $\bigcup_{k=1}^{K} a_k = S$. We denote the set of all partitions of $[n]$ by $\mathscr{A}_{[n]}$, and the set of partitions of $[n]$ with exactly K blocks by $\mathscr{A}_{[n]}^K$, so that $\mathscr{A}_{[n]} = \bigcup_{k=1}^{n} \mathscr{A}_{[n]}^k$. Note that the elements of a partition are unordered, but can be given a canonical ordering by their least elements, and we will refer to the individual blocks using this ordering (where ordering is relevant). A random partition A of $[n]$ is called exchangeable if its probability only depends on the

[11] Aldous [1985] first describes this metaphor, but attributes the idea of the "Chinese restaurant process" metaphor to Jim Pitman. The one-parameter version with $d = 0$ describes the partition structure arising from the Dirichlet process. We will drop the qualification "two-parameter" in the sequel.

[12] Note that if H has atoms it is possible for distinct θ_j to have the same value, and in that case we need to be careful to distinguish the concepts "X_j arising from the same θ_i" and "X_j having the same value". The partition structure captured by the CRP corresponds to the former concept.

number of blocks k and the sizes the blocks, i.e.

$$P(A = \{a_1, \ldots, a_k\}) = f(|a_1|, \ldots, |a_k|) \tag{2.23}$$

for some symmetric function f of *integer compositions* of n (sequences of positive integers with sum n), which is called the *exchangeable partition probability function* (EPPF).[13]

The sequential generative process that gives rise to the two-parameter CRP can be described figuratively as a story in which customers sequentially enter a Chinese restaurant with an "infinite number of tables", where the integers are referred to as *customers* and the blocks as *tables* in the restaurant:

> Customer 1 sits at a table; subsequently, the $(c+1)$-th customer either joins a table which already has customers sitting at it or sits by herself at a new table. The probabilities for these events are
>
> $$\frac{\alpha + |A_c| d}{\alpha + c} \quad \text{for sitting at a new table} \tag{2.24a}$$
> $$\frac{|a| - d}{\alpha + c} \quad \text{for joining an occupied table } a \in A_c \tag{2.24b}$$
>
> where $A_c \in \mathscr{A}_{[c]}$ is the *seating arrangement* (partition) of customers around tables after c customers have entered the restaurant.

Multiplying the probabilities for every customer in the restaurant (or every integer in the partition) together, we obtain a distribution over $\mathscr{A}_{[c]}$ [Pitman, 1996, Theorem 25] which we denote by $\mathrm{CRP}_c(A|\alpha, d)$. For each $A \in \mathscr{A}_{[c]}$,

$$\mathrm{CRP}_c(A|\alpha, d) = \frac{[\alpha + d]_d^{|A|-1}}{[\alpha + 1]_1^{c-1}} \prod_{a \in A} [1 - d]_1^{|a|-1} \tag{2.25}$$

where $[y]_\delta^n = \prod_{i=0}^{n-1}(y + i\delta)$ is *Kramp's symbol* (a rising factorial with increment δ, see Section A.1). The denominator is a normalization constant and does not depend on A. Note that although the distribution only depends on the sizes of the subsets of the partition and the number of subsets (and hence corresponds to the EPPF for this construction), it is normalized over the set $\mathscr{A}_{[c]}$, i.e. $\sum_{A \in \mathscr{A}_{[c]}} P(A) = 1$.

Fixing the number of tables (i.e. the number of blocks in the partition) to be $t \leq c$, the conditional distribution, denoted as $\mathrm{CRP}_{ct}(A|d)$, becomes:

$$\mathrm{CRP}_{ct}(A|d) = \frac{\prod_{a \in A} [1 - d]_1^{|a|-1}}{S_d(c, t)} \quad \text{for each } A \in \mathscr{A}_{[c]}^t, \tag{2.26}$$

[13] Note that $P(A)$ is normalized over the of partitions of $[n]$, $\mathscr{A}_{[n]}$, but f is *not* normalized over the set of compositions of n.

where the normalization constant

$$S_d(c,t) = \sum_{A \in \mathscr{A}^t_{[c]}} \prod_{a \in A} [1-d]_1^{|a|-1} \tag{2.27}$$

is a *generalized Stirling number* of type $(-1,-d,0)$ [Hsu and Shiue, 1998]. These can be computed recursively [Teh, 2006a] in $O(ct)$ time (see Section A.2 in the appendix for details). Note that conditioning on a fixed t the probability of a seating arrangement does not depend on α, only on d.

The generalized Stirling numbers also appear in the distribution of the number of blocks $K_c = |A|$ in CRP partition $A \sim \mathrm{CRP}_c(\alpha, d)$ [Pitman, 2002, Eq. (3.11)]:

$$\mathrm{P}(K_c = t) = \frac{[\alpha+d]_d^{t-1}}{[\alpha+1]_1^{c-1}} S_d(c,t) \tag{2.28}$$

which can be obtained from (2.25) by summing over $A \in \mathscr{A}^t_{[c]}$ using (2.27).

As mentioned before, the connection between the PYP and two-parameter CRP is that the CRP describes the clustering structure of draws from a PYP-distributed random distribution. Consider the model

$$G \sim \mathrm{PY}(\alpha, d, H) \tag{2.29a}$$

$$X_i \mid G \overset{\text{iid.}}{\sim} G \qquad i = 1,\ldots,n \tag{2.29b}$$

The marginal distribution on $X_{1:n}$ can equivalently be described as [Pitman, 2002]:

$$A \sim \mathrm{CRP}_n(\alpha, d) \tag{2.30a}$$

$$\theta_k \mid A \overset{\text{iid.}}{\sim} H \qquad k = 1,\ldots,|A| \tag{2.30b}$$

$$X_i = \varphi_{A,\theta_{1:|A|}}(i) \qquad i = 1,\ldots,n \tag{2.30c}$$

where $\varphi_{A,\theta_{1:|A|}}(\cdot)$ maps an index $i = 1,\ldots,n$ to the θ_k that is associated with the block a_k that contains i.[14] We refer to the combination of the partition $\{a_k\}_{k=1,\ldots,|A|}$ and the associated draws $\theta_k \sim H$ as a *labeled partition*. In the Chinese restaurant metaphor θ_k is referred to as the *dish* that is served on table k. In other words, drawing n observations from a PYP-distributed random distribution G is equivalent to first drawing a random partition of $[n]$ from a CRP (with the same parameters α and d), labelling each of the blocks with an

[14] Note that this can be viewed as a special case of a DP/PYP mixture model: $G \sim PYP(\alpha,d,H), \phi_i|G \sim G, X_i|\phi_i \sim f(\cdot|\phi_i)$, where the likelihood function f is a delta function centered at the parameter value ϕ_i. MCMC and variational inference in such mixture models has been well-studied (see e.g. [Neal, 1998; Blei and Jordan, 2006]), where the inferential challenges are mostly due to the latent variables ϕ_i which need to be inferred together with the clustering structure A or marginalized out.

iid. draw from H, and finally setting the i-th observation equal to the label of the block that contains i.

The representation of the marginal distribution of $X_{1:n}$ in (2.29) in terms of the CRP (2.30) is particularly useful if the goal is making predictions about a new observation X_{n+1}. The predictive distribution over X_{n+1} conditioned on $X_{1:n}$, $A = a_{1:K}$, and $\theta_{1:K}$ is given by [Pitman, 1996]:

$$P(X_{n+1} = s) = \sum_k \frac{|a_k| - d}{n+\alpha} \delta_{\theta_k}(s) + \frac{\alpha + |A|d}{n+\alpha} H(s). \tag{2.31}$$

2.3.3 Customers, Tables, Dishes, and Sections

In what follows we are going to assume that H is discrete with support Σ, so that multiple draws θ_k from H can have the same value. This is in contrast to the more frequently studied case (e.g. [Pitman, 1996, 2002; Neal, 1998, 2000]) where H is assumed to be non-atomic, so that repeated draws are guaranteed to be distinct. In fact, the main inferential problem in our setting is inferring the posterior distribution over the number of blocks that have the same label $s \in \Sigma$. This setting where H is discrete has also been studied by Buntine and Hutter [2012], and arises naturally in hierarchical models where H itself is random and a draw from a PYP (and hence atomic, even if its base distribution is not).

Let us introduce some notation that will be used in description of most inference algorithms that will be discussed in this thesis. Suppose we wish to perform inference in the model (2.29) given observation $x_{1:n}$ of $X_{1:n}$. In the language of Chinese restaurants, this is equivalent to conditioning on the dishes that each customer is served. Since a customer i can only be in the same block with other customers that share the same observed value x_i, the observations split the partition into *sections* (groups of blocks) based on their values. There can be more than one block in each section since multiple blocks can serve the same dish θ_k (due to the discreteness of H). For a given labeled partition $\{(a_k, \theta_k)\}_{k=1,\ldots,K}$ where the a_k form a partition of $[c]$ and the $\theta_k \in \Sigma$ are the labels, let us denote by c_s the number of customers served dish s, i.e. $c_s = \sum_{k:\theta_k=s} |a_k| = \sum_{i=1}^N \mathbf{1}[x_i = s]$, by t_s the number of tables serving dish s, i.e. $t_s = \sum_{k:\theta_k=s} 1$, and let $t = \sum_s t_s = K$ denote the total number of tables/blocks. Further, we denote by c_{sl} the number of customers on the l-th table with that label (after labeling the tables in each section $l = 1, \ldots, L_k$ in the order of least elements), and $A_s \in \mathscr{A}_{c_s t_s}$ the seating arrangement of customers around the tables serving dish s (we re-index the c_s customers to be $[c_s]$). The number of tables t_s serving a particular dish/type s have also been referred to as the *multiplicity* of s by Buntine and Hutter [2012, Definition 32]. In the

discrete setting considered here, these latent multiplicities are the main target of inference, as the assignment of observations to sections is given and the partitions A_s are relevant for making predictions using (2.31) only through the size of the partition $|A_s| = t_s$. Using these definitions, the joint distribution over seating arrangements *and* observations can then be written as

$$P(\{c_s, t_s, A_s\}, x_{1:n}) = \left(\prod_{s\in\Sigma} G_0(s)^{t_s}\right)\left(\frac{[\alpha+d]_d^{t_\cdot-1}}{[\alpha+1]_1^{c_\cdot-1}} \prod_{s\in\Sigma}\prod_{a\in A_s} [1-d]_1^{|a|-1}\right), \tag{2.32}$$

where $t_\cdot = \sum_{s\in\Sigma} t_s$ and $c_\cdot = \sum_{s\in\Sigma} c_s = n$.[15] We can marginalize out the seating arrangement $\{A_s\}$ from (2.32) using (2.27):

$$P(\{c_s, t_s\}, x_{1:n}) = \left(\prod_{s\in\Sigma} G_0(s)^{t_s}\right)\left(\frac{[\alpha+d]_d^{t_\cdot-1}}{[\alpha+1]_1^{c_\cdot-1}} \prod_{s\in\Sigma} S_d(c_s, t_s)\right). \tag{2.33}$$

To simplify notation further on, it is convenient to define

$$f^{\mathrm{CT}}_{\alpha,d}(\mathbf{c}, \mathbf{t}) = \frac{[\alpha+d]_d^{t_\cdot-1}}{[\alpha+1]_1^{c_\cdot-1}} \prod_{s\in\Sigma} S_d(c_s, t_s), \tag{2.34}$$

where $\mathbf{c}$ and $\mathbf{t}$ are $\{c_s\}$ and $\{t_s\}$ stacked into vectors, so that $P(\{c_s, t_s\}, x_{1:n}) = f^{\mathrm{CT}}_{\alpha,d}(\mathbf{c}, \mathbf{t}) \prod_{s\in\Sigma} G_0(s)^{t_s}$. We can also re-write the predictive distribution (2.31) (conditioned on the previous c observations with counts c_s, and the multiplicities t_s derived from the seating arrangement) using this notation to obtain

$$\mathrm{P}(X_{c+1} = s | \{c_s, t_s\}) = \frac{c_s - t_s d}{c+\alpha} + \frac{\alpha + t d}{c+\alpha} H(s). \tag{2.35}$$

This expression already reveals part of the relationship between the hierarchical PYP language model (Section 2.4) and Kneser-Ney smoothing (Section 2.2.3): As Teh [2006a] pointed out, setting $\alpha = 0$ and $t_s = \mathrm{I}[c_s > 0]$ in (2.35) yields the absolute discounting formula (2.15).

2.3.4 Power-Law Properties of the Pitman-Yor Process

One of the main reasons for choosing the Pitman-Yor process over the in many respects simpler Dirichlet process (or a finite Dirichlet distribution) as a prior for discrete distributions is that it matches the power-law behavior found in many physical processes and in natural language (see Figure 2.5 and Section 2.1).

There are in fact several types of power-law scaling present in the Pitman-Yor process: The first is that the probability of observing a table of size k scales

[15] We have omitted the set subscript $\{\cdot\}_{s\in\Sigma}$. We will drop these subscripts when they are clear from context. Note that this distribution is also over the observations $x_{1:n}$, not just the counts c_s, and that there are $\frac{c!}{\prod_s c_s!}$ possible observation sequences for a given set of counts $\{c_s\}$.

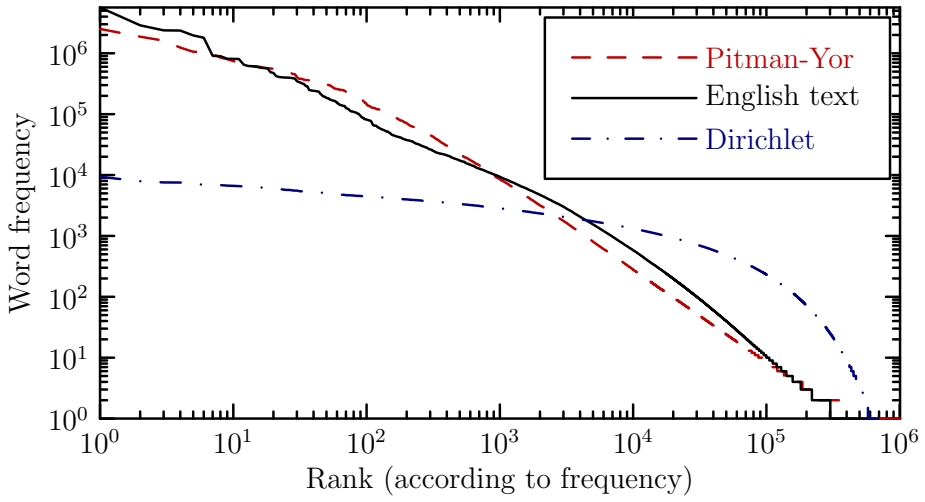


Figure 2.5: Illustration of the type/word frequency power law comparing samples from the Dirichlet process and the Pitman-Yor process with the empirical word frequencies in a large corpus of English text (a version of the Wikipedia corpus). For the DP and PYP we plot the block sizes a_k for the corresponding CRP partition of N customers, where N is the size of the corpus and the parameters α and d (for the PYP) were optimized to maximize the likelihood. The power law scaling of the PYP is a much better fit to the empirical frequencies.

as $k^{-(1+d)}$ as the number of customers gets large. A directly related power law property is that the relative size of the k-th largest table (or equivalently the k-th largest stick-breaking weight) as $n \to \infty, k \to \infty$ follows a power-law with index $1/d$ [Pitman, 2002, Lemma 3.11]. These two properties are shown in figures 2.6 and 2.7.

Another power-law property is that under the two-parameter CRP prior, the expected number of tables grows as a power-law with index $0 < d < 1$ with the number of customers, i.e. [Pitman, 2002, Section 3.3]:

$$E_{\alpha,d}[K_n] \sim \frac{\Gamma(\alpha+1)}{d\Gamma(\alpha+d)} n^d . \tag{2.36}$$

This is illustrated in Figure 2.8.

As pointed out by Gnedin et al. [2007], these asymptotic power-law properties are equivalent, and are not unique to the Pitman-Yor process, but hold more generally for the wider class of σ-stable processes. The article also describes further power law properties, e.g. that number of blocks of a particular size scales as n^d as $n \to \infty$.

An interesting avenue for further research is to analyze the power-law properties of wider classes of non-parametric priors and how they relate to each other. Of particular interest is also the non-asymptotic behavior and the study of more flexible priors which can more closely mimic the properties observed in real data, e.g. the two different power law regimes for low- and high-frequency

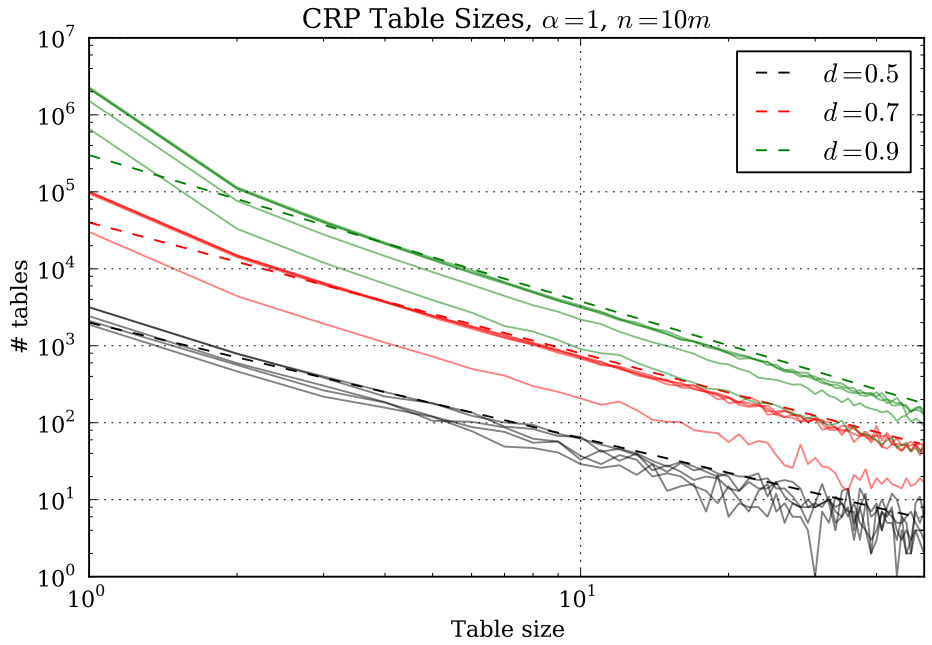


Figure 2.6: Samples from the two-parameter CRP with $\alpha = 1$ and 10 million customers. Shown are 5 samples (solid lines) of the resulting numbers of tables of a particular size, i.e. $|\{k : |a_k| = j\}|$ as a function of the table size j. For comparison, the power law $\propto j^{-(1+d)}$ is also shown (dashed line).

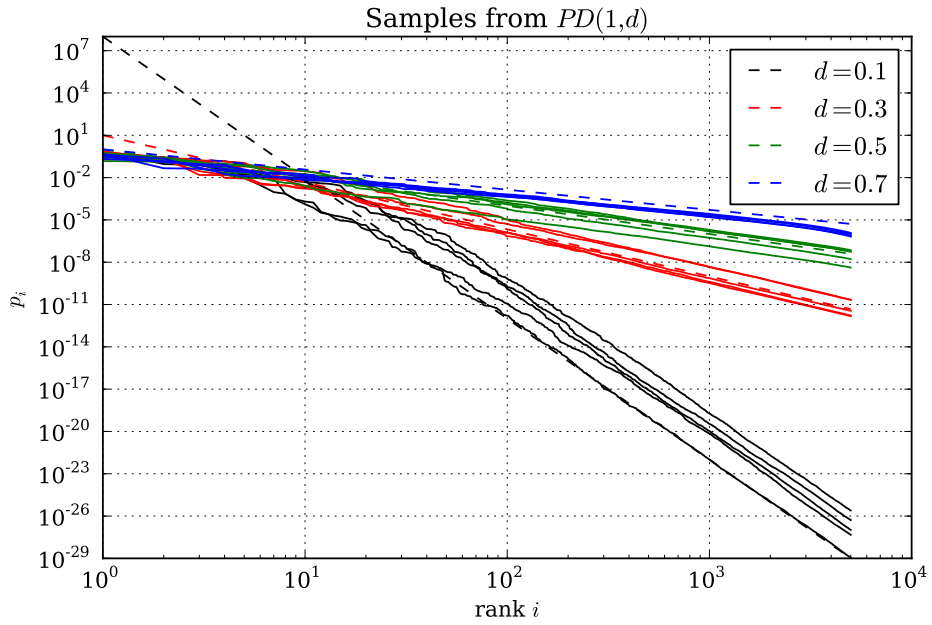


Figure 2.7: Samples from the Poisson-Dirichlet distribution (i.e. the weights of the Pitman-Yor process ranked by decreasing size) for $\alpha = 1$ and four different values for d. Also shown for comparison as dashed lines are scaled versions of $i^{-\frac{1}{d}}$, matching the slopes of the tails.

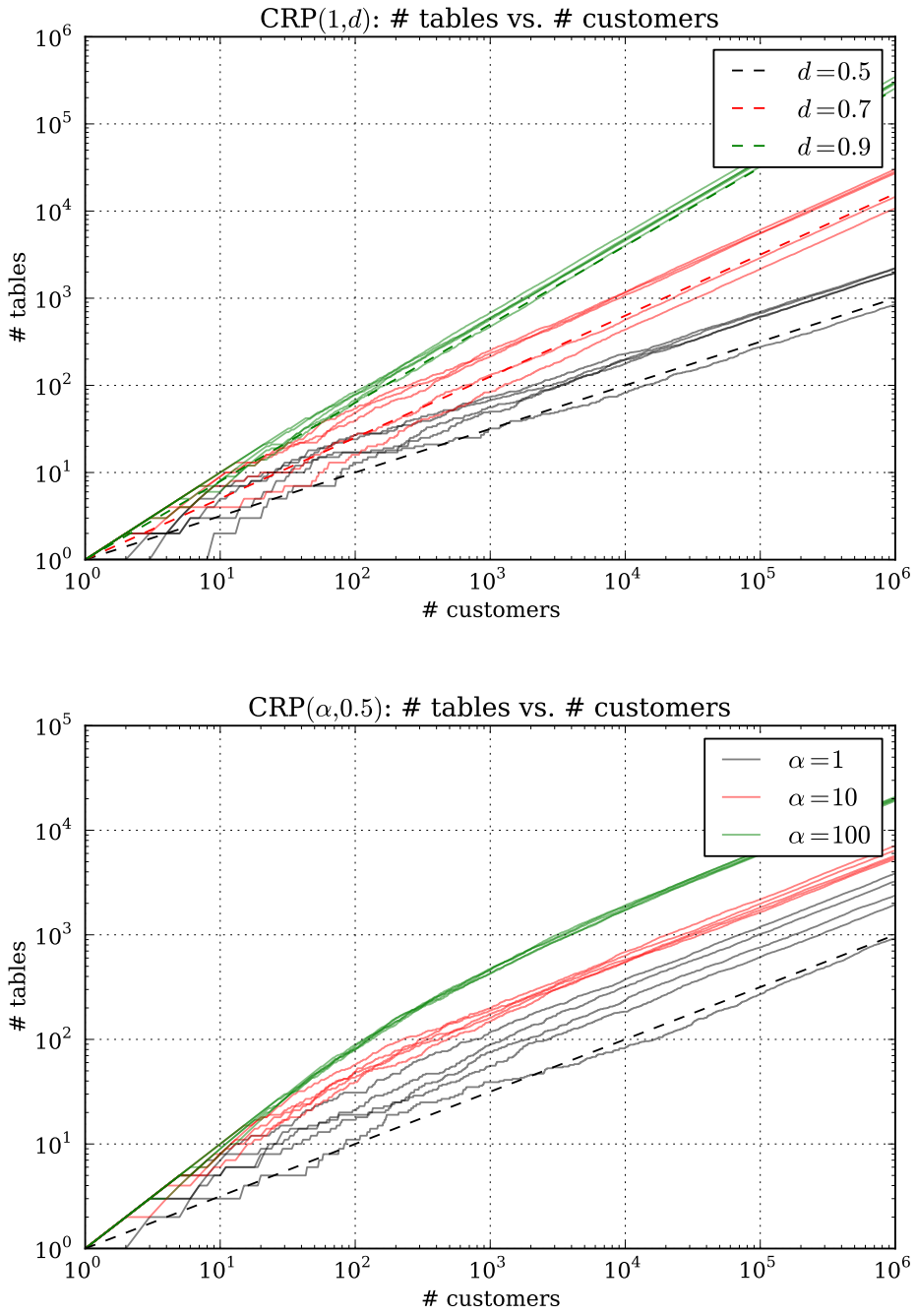


Figure 2.8: Number of tables t in a sample from the two-parameter CRP(α, d) as the number of customers c increases from 1 to 1 million. Shown are 5 samples (solid lines) for different d (top) and α (bottom). For comparison, the limiting the power laws $\propto c^d$ are also shown (dashed lines). Note that in the bottom plot, α does not influence the asymptotic scaling but does affect the initial growth as well as the variance.

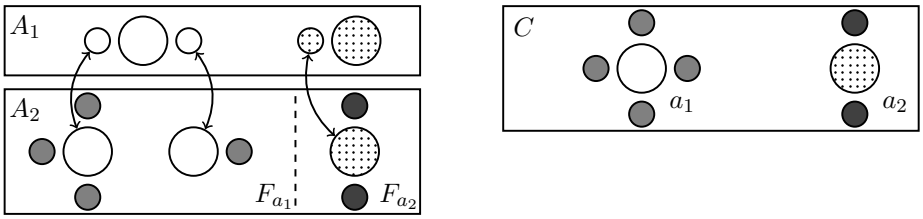


Figure 2.9: Illustration of the relationship between the restaurants A_1, A_2, C and F_a in Theorem 2 for $c = 6$ and $|A_2| = 3$. Tables in A_2 correspond to customers in A_1, and C is obtained by merging the tables in A_2 that correspond to customers sitting at the same table in A_1.

words seen in Figure 2.5. Such "double power-law" behavior has recently been considered by Ayed et al. [2019].

2.3.5 Coagulation and Fragmentation

One property of the Pitman-Yor process that has been essential to the development of the Sequence Memoizer model that will be described in Chapter 4 is that, under certain conditions, a draw from a Pitman-Yor process with a Pitman-Yor-distributed base measure is itself marginally Pitman-Yor distributed. This property allows "chains" of Pitman-Yor distributed random measures to be marginalized out analytically. More precisely, that main result in this regard is:

Theorem 1 (adapted from [Pitman, 1999; Ho et al., 2006])**.** If $G_1 \sim \mathrm{PY}(\alpha/d_2, d_1, H)$ and $G_2 \mid G_1 \sim \mathrm{PY}(\alpha, d_2, G_1)$, then marginally $G_2 \sim \mathrm{PY}(\alpha, d_1 d_2, H)$ for $0 < d_1 < 1$, $0 < d_2 < 1$, $-d_1 < \alpha/d_2$.

This result was first obtained by Pitman [1999] and can be understood both in terms of the CRP (i.e. via the partition structure, as in [Pitman, 1999]) and in terms of the Poisson-Dirichlet distribution (as in [Ho et al., 2006]).

Viewed in terms of partitions distributed according to a CRP, the coagulation and fragmentation operations can be understood as follows (see Figure 2.9 for an illustration): Let $c \geq 1$ and suppose $A_2 \in \mathscr{A}_c$ and $A_1 \in \mathscr{A}_{|A_2|}$ are two seating arrangements where the number of customers in A_1 is the same as that of tables in A_2. Each customer in A_1 can be put in one-to-one correspondence to a table in A_2 and sits at a table in A_1. Now consider re-representing A_1 and A_2. Let $C \in \mathscr{A}_c$ be the seating arrangement obtained by coagulating (merging) tables of A_2 corresponding to customers in A_1 sitting at the same table. Further, split A_2 into sections, one for each table $a \in C$, where each section $F_a \in \mathscr{A}_{|a|}$ contains the $|a|$ customers and tables merged to make up a. The converse of coagulating tables of A_2 into C is of course to fragment each table $a \in C$ into the smaller tables in F_a. Note that there is a one-to-one correspondence between

tables in C and in A_1, and the number of customers in each table of A_1 is that of tables in the corresponding F_a. Thus A_1 and A_2 can be reconstructed from C and $\{F_a\}_{a\in C}$.

Theorem 2 (Pitman [1999]; Ho et al. [2006])**.** Suppose $A_2 \in \mathscr{A}_c$, $A_1 \in \mathscr{A}_{|A_2|}$, $C \in \mathscr{A}_c$ and $F_a \in \mathscr{A}_{|a|}$ for each $a \in C$ are related as above. Then the following describe equivalent distributions:
(I) $A_2 \sim \mathrm{CRP}_c(\alpha d_2, d_2)$ and $A_1|A_2 \sim \mathrm{CRP}_{|A_2|}(\alpha, d_1)$.
(II) $C \sim \mathrm{CRP}_c(\alpha d_2, d_1 d_2)$ and $F_a|C \sim \mathrm{CRP}_{|a|}(-d_1 d_2, d_2)$ for each $a \in C$.

Proof. We simply show that the joint distributions are the same. Starting with (I) and using (2.25),

$$\begin{aligned} P(A_1,A_2) &= \left(\frac{[\alpha+d_1]_{d_1}^{|A_1|-1}}{[\alpha+1]_1^{|A_2|-1}}\prod_{a\in A_1}[1-d_1]_1^{|a|-1}\right)\left(\frac{[\alpha d_2+d_2]_{d_2}^{|A_2|-1}}{[\alpha d_2+1]_1^{c-1}}\prod_{b\in A_2}[1-d_2]_1^{|b|-1}\right) \\ &= \frac{[\alpha d_2+d_1d_2]_{d_1d_2}^{|A_1|-1}}{[\alpha d_2+1]_1^{c-1}}\left(\prod_{a\in A_1}[d_2-d_1d_2]_{d_2}^{|a|-1}\right)\left(\prod_{b\in A_2}[1-d_2]_1^{|b|-1}\right). \end{aligned}$$

We used the identity $[\beta\delta+\delta]_\delta^{n-1} = \delta^{n-1}[\beta+1]_1^{n-1}$ for all β,δ,n. Re-grouping the products and expressing the same quantities in terms of C and $\{F_a\}$,

$$= \frac{[\alpha d_2+d_1d_2]_{d_1d_2}^{|C|-1}}{[\alpha d_2+1]_1^{c-1}}\prod_{a\in C}\left([d_2-d_1d_2]_{d_2}^{|F_a|-1}\prod_{b\in F_a}[1-d_2]_1^{|b|-1}\right) = P(C,\{F_a\}_{a\in C}).$$

We see that conditioning on C each $F_a \sim \mathrm{CRP}_{|a|}(-d_1d_2, d_2)$. Marginalizing $\{F_a\}$ out using (2.25),

$$P(C) = \frac{[\alpha d_2+d_1d_2]_{d_1d_2}^{|C|-1}}{[\alpha d_2+1]_1^{c-1}}\prod_{a\in C}[1-d_1d_2]_1^{|a|-1}.$$

So $C \sim \mathrm{CRP}_c(\alpha d_2, d_1 d_2)$ and *(I)*⇒*(II)*. Reversing the same argument shows that *(II)*⇒*(I)*. □

Statement *(I)* of the theorem is exactly the *Chinese restaurant franchise* ([Teh et al., 2006]; see also Sec. 2.4 below) of the hierarchical model $G_1|G_0 \sim \mathrm{PY}(\alpha, d_1, G_0)$, $G_2|G_1 \sim \mathrm{PY}(\alpha d_2, d_2, G_1)$ with c iid draws from G_2. The theorem shows that the clustering structure of the c customers in the franchise is equivalent to the seating arrangement in a CRP with parameters $\alpha d_2, d_1 d_2$, i.e. $G_2|G_0 \sim \mathrm{PY}(\alpha d_2, d_1 d_2, G_0)$ with G_1 marginalized out. Conversely, the fragmentation operation *(II)* regains Chinese restaurant representations for both $G_2|G_1$ and $G_1|G_0$ from one for $G_2|G_0$.

2.3.6 Inference in Basic PYP Models

Given observations $X_{1:n} = x_{1:n}$ from the single PYP model given by Equation (2.29) one may be interested in either making predictions by computing the predictive probability of a new observation, or in computing the posterior distribution over the latent variables. The latter is typically used as a means to achieve the former, as the predictive distribution is computed as an expectation of (2.31) (or (2.35)) under the posterior distribution of the latent variables. In the CRP representation (2.30), i.e. when the random measure G is integrated out, posterior inference amounts to computing or approximating the posterior distribution over seating arrangements A or the multiplicities t_s given the observations $x_{1:n}$, or equivalently, given the symbol occurrence counts $\{c_s\}_{s\in\Sigma}$.[16] Inferring only the multiplicities t_s is sufficient if one is only interested in making predictions by averaging the predictive distribution (2.35), which does not depend on the exact seating arrangement.[17] Computing the posterior distribution over seating arrangements by explicit enumeration and re-normalization of (2.32) is prohibitively expensive, as the number of partitions (given by the Bell number) grows super-exponentially. Computing the posterior distribution over t_s by enumerating $t_s = 1,\ldots,c_s$ and computing and normalizing (2.33) is feasible for small problems, but the number of possible $\{t_s\}$, given by $\prod_{s\in\Sigma:c_s>1} c_s$, also grows quickly.[18] The posterior distribution over $\{t_s\}$ for various hyperparameter settings is illustrated in Appendix A.3.

Basic Gibbs Sampler For A Single PYP

Deriving a basic Gibbs sampler for the simple model (2.29) where observations are directly drawn from a PYP-distributed random measure which is integrated out, yielding model (2.30), is straight-forward and can be seen as a special case of the collapsed Gibbs sampler proposed for DP mixture models in [Neal, 1992] (Algorithm 3 in the classification of [Neal, 1998]) for the case where the likeli-

[16] Note that the necessity for inference in this simple model stems solely from the fact that we are considering a base distribution H that is discrete: If H has no atoms, the predictive distribution is directly given by (2.31) with all quantities fixed by the observations. In this case, the posterior PYP $G \mid x_{1:n}$ can also be given an explicit characterization as a mixture of point masses at the observed values and a PYP with modified parameters [Pitman, 1996] (see also [Teh and Jordan, 2010]).

[17] Additionally, one could place priors on the parameters α and d and infer (or average over) their posterior distribution. We will focus on the case with fixed α and d here, but in the hierarchical models described in later chapters these parameters will either be inferred using MCMC or optimized via other means. See e.g. [Carlton, 1999, Chapter 5] for a description of several strategies for estimating the parameters in the single PYP case.

[18] The Stirling numbers $S_d(c,t)$ that appear in (2.33) can be computed recursively in $O(c*t)$ for all $c' \le c$, $t' \le t$ (see Section A.2), so in practice it often makes sense to pre-tabulate these numbers.

hood is a delta function at the parameter value. The target of the sampler is the posterior distribution of the partition A. It proceeds by in turn removing each $i = 1,\ldots,n$ from the block a_{k^*}, which currently contains i and then re-assigning it to one of the blocks according to their conditional posterior probability. As the indices $i = 1,\ldots,n$ are exchangeable under the CPR prior, we can easily compute the posterior probability of customer i either joining an existing block a_k or creating a new block a_{K+1} (where K is the current number of blocks) by treating it as the last customer, so that $P\left(\text{customer } i \text{ joining block } k \mid A^{\neg i}\right)$

$$\propto \begin{cases} c_k - d & \text{if } k = 1,\ldots,K \wedge \forall j \in a_k : x_j = x_i \\ (\alpha + dK)H(x_i) & \text{if } k = K+1 \end{cases} \tag{2.37}$$

where $A^{\neg i}$ denotes the the current partition with customer i removed and c_k denotes the number of (remaining) customers in block k. In other words, the removed customer i joins an already existing block with the same label (i.e. where all the other $x_j = x_i$) with probability proportional to the number of customers sitting there discounted by d (i.e. $\propto c_k - d$), and starts a new block with probability proportional to $(\alpha + dK)$ times the probability of x_i under the base distribution $H(x_i)$.[19] This simple remove-add Gibbs sampler (summarized in Algorithm 1), which sequentially removes customers from the current seating arrangement and then re-inserts them according to their conditional posterior probability forms the basis for the Gibbs samplers used for performing inference in the hierarchical PYP models discussed next, where the remove and add steps are applied recursively throughout the hierarchy.

[19] Note that this procedure does not depend on the labeling of the blocks (or the customers), only on the sizes of the resulting blocks, so that implementations need not re-label blocks or customers.

Algorithm 1 Basic remove/add Gibbs sampler for the posterior distribution over partitions for the model in (2.30). The utility function DRAWPROPORTIONAL samples an index from the discrete distribution proportional to the provided weights (e.g. using the inverse CDF method).

function REMOVECUSTOMER($a_{1:K}, i$)
  Remove Customer i from seating arrangement
  return $\{a_1, \ldots, a_{k^*} \setminus i, \ldots, a_K\}$ ▷ k^* denotes block containing i
end function

function ADDCUSTOMER($a_{1:K}, x_{1:n}, i$)
  Add i to seating arrangement $a_{1:K}$ according to posterior distribution
  for $k = 1, \ldots, K$ **do**
    $p_k \leftarrow (|a_k| - d) \times \mathbf{1}[\forall j \in a_k : x_i = x_j]$
  end for
  $p_{K+1} \leftarrow (\alpha + dK) H(x_i)$
  $\tilde{k} \leftarrow$ DRAWPROPORTIONAL($p_1, \ldots, p_{K+1}$)
  if $\tilde{k} \leq K$ **then**
    $\tilde{a}_{\tilde{k}} \leftarrow a_k \cup \{i\}$ ▷ Join existing block
  else
    $\tilde{a}_{K+1} = \{i\}$ ▷ Create new block
  end if
  return $\{a_1, \ldots, \tilde{a}_{\tilde{k}}, \ldots, a_K\}$
end function

function REMOVEADDSAMPLER($a^{(0)}_{1:K^{(0)}}$)
  Obtain samples $A^{(j)} = (a^{(j)}_1, \ldots, a^{(j)}_{K^{(j)}})$, $j = 1, \ldots, \#samples$
  for $j = 1, \ldots,$ # samples **do**
    for $i = 1, \ldots, N$ **do**
      $a^{(j)\setminus i}_{1:K^{(j-1)}} \leftarrow$ REMOVECUSTOMER($a^{(j-1)}_{1:K^{(j-1)}}, i$)
      $a^{(j)}_{1:K^{(j)}} \leftarrow$ ADDCUSTOMER($a^{(j)\setminus i}_{1:K^{(j-1)}}, i$)
    end for
  end for
  return $\left(A^0, \ldots, A^{\#\text{samples}}\right)$
end function

2.4 The Hierarchical Pitman-Yor Process

Bayesian hierarchical modelling is a powerful framework for expressing dependencies between (latent) random variables, where the dependencies are induced by (recursively) placing a shared prior distribution on the parameters of the distributions of the dependent random variables (see e.g. [Gelman et al., 2004, Chapter 5] for a general introduction and [Teh and Jordan, 2010] for an introduction to the nonparametric Bayesian setting).

A *hierarchical Pitman-Yor process* (HPYP) is such a hierarchical Bayesian model, where the latent random variables in the hierarchy are random distributions which are given a Pitman-Yor process prior with a shared base distribution.

As an illustration, let us first consider a simple example of such a hierarchical Pitman-Yor process model:

$$G_0 \sim \mathrm{PY}(a_0, d_0, \mathscr{U}_\Sigma) \tag{2.38a}$$

$$G_1 \sim \mathrm{PY}(a_1, d_1, G_0) \tag{2.38b}$$

$$G_2 \sim \mathrm{PY}(a_2, d_2, G_0) \tag{2.38c}$$

$$x_i^{(1)} \overset{\text{iid.}}{\sim} G_1 \qquad i = 1, \ldots, n_1 \tag{2.38d}$$

$$x_i^{(2)} \overset{\text{iid.}}{\sim} G_2 \qquad i = 1, \ldots, n_2 \tag{2.38e}$$

where $\mathscr{U}_\Sigma$ is the uniform distribution on the set Σ with PMF $p(s) = 1/|\Sigma|$ for all $s \in \Sigma$. The model has a hierarchical structure, in that the two distributions G_1 and G_2 from which observations are drawn share a common latent base distribution G_0, which is itself PYP distributed with a fixed, uniform base distribution. Intuitively, G_0 models common behavior of both G_1 and G_2, allowing for sharing of statistical strength between the two, so that they can be estimated more accurately from less data.

Another way to view this model is through a hierarchical generalization of the Chinese restaurant process known as the *Chinese restaurant franchise* (CRF) introduced by Teh et al. [2006]. In the CRF representation of a model like (2.38), all random distributions G_0, G_1, G_2 are integrated out and replaced by a Chinese restaurant in which customers are seated according to a CRP. As dishes served on the tables in the G_1 and G_2 restaurants correspond to draws from their base distribution G_0, these correspond to customers entering the restaurant associated with G_0. In other words, whenever a customer entering the G_1 or G_2 restaurant sits at a new table, a customer enters the G_0 restaurant (and may or may not sit at a new table there). By using this idea that each dish/table at a lower level corresponds to a customer at the next higher level, this process can straightforwardly be extended to deeper and wider hierarchies.

2.4.1 The Hierarchical Pitman-Yor Process Language Model

A language model can be viewed as collection of distributions $\{G_{\mathbf{u}}(s)\}_{\mathbf{u}}$ over the next word s given a *context* of previous words $\mathbf{u}$. The probability of a sequence of words $x_{1:T}$ can then be computed as

$$\prod_{t=1}^{T} \mathrm{P}(x_t | x_{1:t-1}) = \prod_{t=1}^{T} G_{c(x_{1:t-1})}(s) \tag{2.39}$$

where $c(\cdot)$ maps the history to a context in the set of possible contexts, e.g. a truncation to the K preceding symbols in the case of a Markov model of order $K+1$.

This collection of distributions can then be given a hierarchical prior similar to the one in (2.38), thus allowing statistical strength to be shared between different contexts, allowing the $G_{\mathbf{u}}$ to be estimated accurately in the presence of data sparsity.

One of the first applications hierarchical Bayesian modelling to the language modelling problem is the Hierarchical Dirichlet Language Model (HDLM) of MacKay and Bauman Peto [1995], which is an ancestor of the (substantially more complex) models described here. The HDLM is a bigram language model, where the conditional distributions $G_s(\cdot) \stackrel{\text{def.}}{=} \mathrm{P}(x_t|x_{t-1} = s)$ of a word following a context word s are given Dirichlet prior distributions $G_s \sim \mathrm{Dir}(\alpha\mathbf{m})$, whose shared parameters $\alpha\mathbf{m}$ are given a shared prior, thus resulting in a hierarchical model with two levels.[20]

MacKay and Bauman Peto [1995] developed this model to provide a Bayesian alternative to the traditional ad-hoc smoothing and interpolation techniques described in Section 2.2.2. The resulting predictive distribution can be seen as a context-adaptive interpolation between the relative frequency of the bigram and a context-dependent smoothing vector. The main advantages of this model compared to the ad-hoc techniques are that it is based on a well-specified model which makes prior assumptions explicit, and that by inferring all parameters using Bayesian inference no model selection techniques like cross-validation are required.

At a high level, the hierarchical Pitman-Yor process language model (HPYPLM) introduced by Teh [2006a,b] can be seen as an extension to this model in two directions: the Dirichlet distribution is replaced with a Pitman-Yor process and the two-level hierarchy of the HDLM bigram model is extended to a general n-gram hierarchy of some fixed depth K. The former extension allows for more accurate smoothing, as the Pitman-Yor process is a better fit to the power-law properties observed in natural language (as discussed in Sections 2.1.1 and 2.3.4), while the latter extension allows the model to make use of longer contexts to make predictions. On the downside, inference in this class of models is more involved and requires approximate inference techniques.

As before, let Σ be the discrete set of symbols making up the sequences to be modeled, and let Σ^K be the set of length K sequences of symbols from Σ. The HPYP models a sequence $x_{1:T} = x_1, x_2, \ldots, x_T$ using a set of conditional

[20] In the HDLM paper the prior on $\alpha\mathbf{m}$ is not given an explicit form but instead $\alpha\mathbf{m}$ is assumed to be a parameter that is determined by maximizing the evidence. While it would be straightforward to use and explicit prior, e.g. a Dirichlet prior on $\mathbf{m}$ and a Gamma prior on α, MacKay and Bauman Peto [1995] choose not to do so to simplify the inference and estimation.

distributions:

$$P(x_{1:T}) = \prod_{t=1}^{T} P(x_t|x_{1:t-1}) = \prod_{t=1}^{T} G_{x_{t-K:t-1}}(x_t), \tag{2.40}$$

where $G_{\mathbf{u}}(s)$ is the conditional probability of the symbol $s \in \Sigma$ occurring after a context $\mathbf{u} \in \Sigma^K$ (the sequence of symbols occurring before s). The parameters of the model consist of all the conditional distributions $\{G_{\mathbf{u}}\}_{\mathbf{u}\in\Sigma^K}$, and are given the following hierarchical Pitman-Yor process (HPYP) prior:

$$\begin{aligned} G_\varepsilon &\sim \mathrm{PY}(\alpha_\varepsilon, d_\varepsilon, H) \\ G_{\mathbf{u}}|G_{\sigma(\mathbf{u})} &\sim \mathrm{PY}(\alpha_{\mathbf{u}}, d_{\mathbf{u}}, G_{\sigma(\mathbf{u})}) \qquad \text{for } \mathbf{u} \in \Sigma^{\leq K}\backslash\{\varepsilon\}, \end{aligned} \tag{2.41}$$

where ε is the empty sequence, $\sigma(\mathbf{u})$ is the sequence obtained by dropping the first symbol in $\mathbf{u}$ (i.e. $\sigma(\mathbf{u})$ denotes the longest *proper suffix* of $\mathbf{u}$), $\Sigma^{\leq K} = \bigcup_{k=1}^{K} \Sigma^k$ is the set of all strings over Σ with length less than or equal to K, and H is a fixed, non-random base distribution over Σ (which is taken to be uniform over a finite Σ in [Teh, 2006a,b] and in this thesis, but could be more complex in general). In order to reduce the number of parameters, the context-dependent discount and concentration parameters $d_{\mathbf{u}}$ and $\alpha_{\mathbf{u}}$ are assumed to be shared between contexts of the same length, i.e. $d_{\mathbf{u}} = d_{|\mathbf{u}|}$ and $\alpha_{\mathbf{u}} = \alpha_{|\mathbf{u}|}$.

Teh [2006a] proposed performing inference in the HPYP language model by marginalizing out the distributions $\{G_{\mathbf{u}}\}$ and representing the hierarchy over the resulting CRPs using a Chinese restaurant franchise [Teh et al., 2006]. In this representation, each $G_{\mathbf{u}}$ has a corresponding restaurant indexed by $\mathbf{u}$.[21] Customers in the restaurant are draws from $G_{\mathbf{u}}$, tables are draws from its base distribution $G_{\sigma(\mathbf{u})}$, and dishes are the drawn values from Σ. For each $s \in \Sigma$ and $\mathbf{u} \in \Sigma^{\leq K}$, let $c_{\mathbf{u}s}$ and $t_{\mathbf{u}s}$ be the numbers of customers and tables in restaurant $\mathbf{u}$ served dish s, and let $A_{\mathbf{u}} \in \mathcal{A}_{c_{\mathbf{u}s}t_{\mathbf{u}s}}$ be their seating arrangement. Each observation of x_t in context $\mathbf{u} = x_{t-K:t-1}$ corresponds to a customer in restaurant $\mathbf{u}$ who is served dish x_t, and each table in a restaurant $\mathbf{u}$—being a draw from the base distribution $G_{\sigma(\mathbf{u})}$—corresponds to a customer in the parent restaurant $\sigma(\mathbf{u})$. Thus, the numbers of customers and tables in the CRF representation of the hierarchy have to satisfy the constraints

$$c_{\mathbf{u}s} = c^x_{\mathbf{u}s} + \sum_{\mathbf{v}:\sigma(\mathbf{v})=\mathbf{u}} t_{\mathbf{v}s}, \tag{2.42}$$

where $c^x_{\mathbf{u}s}$ denotes the number of customers corresponding to the observations, which enter only in the leafs of the hierarchy, i.e. $c^x_{\mathbf{u}s} = \#(\mathbf{u}s)$ if $|\mathbf{u}| = K$ and $c^x_{\mathbf{u}s} = 0$ otherwise. In other words, every customer in each restaurant in the

[21] In the sequel, we will simply refer to this representation of $G_{\mathbf{u}}$ in the CRF as the "restaurant $\mathbf{u}$".

hierarchy either corresponds to an observation x_t or to a table in a lower-level restaurant.

Let $\mathscr{U}$ denote the set of all contexts and let $\pi(\mathbf{u})$ denote the parent of context $\mathbf{u}$ in the hierarchy, so that for the HPYP model we have $\mathscr{U} = \Sigma^{\leq K}$ and $\pi(\mathbf{u}) = \sigma(\mathbf{u})$.[22] The goal of inference is to compute the posterior over the states $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}, A_{\mathbf{u}}\}_{s\in\Sigma,\mathbf{u}\in\mathscr{U}}$ of the restaurants (and possibly the concentration and discount parameters) given the observations $x_{1:T}$, which can then be used to compute the predictive distribution for a new context-symbol pair as an expectation over this posterior. The joint distribution over latent variables and observations can be obtained by multiplying the probabilities of all seating arrangements (2.32) in all restaurants:

$$P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}, A_{\mathbf{u}}\}, x_{1:T}) = \left(\prod_{s\in\Sigma} H(s)^{t_{\varepsilon s}}\right) \prod_{\mathbf{u}\in\mathscr{U}} \left(\frac{[\alpha_{\mathbf{u}} + d_{\mathbf{u}}]_{d_{\mathbf{u}}}^{t_{\mathbf{u}\cdot}-1}}{[\alpha_{\mathbf{u}}+1]_1^{c_{\mathbf{u}\cdot}-1}} \prod_{s\in\Sigma}\prod_{a\in A_{\mathbf{u}}} [1-d_{\mathbf{u}}]_1^{|a|-1} \right). \tag{2.43}$$

The first parentheses contain the probability of draws from the overall base distribution H, and the second parentheses contain the probability of the seating arrangement in restaurant $\mathbf{u}$.

Starting from the joint distribution over the Chinese restaurant franchise (2.43) we can integrate out the seating arrangements $\{A_{\mathbf{u}}\}$ using (2.33), resulting in the joint distribution over $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$:

$$P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}, x_{1:T}) = \left(\prod_{s\in\Sigma} H(s)^{t_{\varepsilon s}}\right) \prod_{\mathbf{u}\in\mathscr{U}} \left(\frac{[\alpha_{\mathbf{u}} + d_{\mathbf{u}}]_{d_{\mathbf{u}}}^{t_{\mathbf{u}\cdot}-1}}{[\alpha_{\mathbf{u}}+1]_1^{c_{\mathbf{u}\cdot}-1}} \prod_{s\in\Sigma} S_{d_{\mathbf{u}}}(c_{\mathbf{u}s}, t_{\mathbf{u}s}) \right). \tag{2.44}$$

Note that each $c_{\mathbf{u}s}$ is in fact determined by (2.42) so that the only unobserved variables in (2.44) are $\{t_{\mathbf{u}s}\}$.

Given a state of the restaurants drawn from the posterior, the conditional predictive probability of symbol s in context $\mathbf{v}$ can then be computed by using (2.31) recursively (with $P^*_{\pi(\varepsilon)}(s)$ defined to be $H(s)$):

$$P^*_{\mathbf{u}}(s|\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}) = \frac{c_{\mathbf{u}s} - t_{\mathbf{u}s} d_{\mathbf{u}}}{\alpha_{\mathbf{u}} + c_{\mathbf{u}\cdot}} + \frac{\alpha_{\mathbf{u}} + t_{\mathbf{u}\cdot} d_{\mathbf{u}}}{\alpha_{\mathbf{u}} + c_{\mathbf{u}\cdot}} P^*_{\pi(\mathbf{u})}(s). \tag{2.45}$$

Note again that this predictive distribution only depends on the state of the restaurants only through the counts $c_{\mathbf{u}s}$ and $t_{\mathbf{u}s}$, but not directly on the partitions $A_{\mathbf{u}}$. In order to make predictions with an HPYP model, the predictive distribution (2.45) is averaged with respect to the distribution over counts / seating arrangements (2.44),

$$P^*_{\mathbf{u}}(s) = \mathbb{E}_{\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}}\left[P^*_{\mathbf{u}}(s|\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\})\right], \tag{2.46}$$

[22] This more general notation is introduced here in anticipation of the discussion of the Sequence Memoizer model in the next chapter, where the underlying hierarchical model is essentially identical, but the base set of contexts and their parent-child relationship is different.

typically using a Monte Carlo approximation to the expectation and averaging the predictive distribution with respect to samples drawn from the posterior distribution.

2.4.2 Relation to Interpolated Kneser-Ney Smoothing

Both Teh [2006a] and Goldwater et al. [2006a] (independently) noted that the HPYP language model can be related to interpolated Kneser-Ney smoothing, in that a particular setting of the counts $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$ yields the same predictive distribution as the one obtained by Kneser-Ney.

As we noted in Section 2.2.3, interpolated KN smoothing combines two ingredients: an interpolation mechanism based on absolute discounting, and an estimator based on modified counts for the lower-order predictors. We have already seen in Section 2.3.3 that the PYP predictive distribution (2.31) is the same as the absolute discounting formula (2.15) for the special case when $\alpha = 0$ and $t_s = \mathbf{1}[c_s > 0]$. Interpolated Kneser-Ney's modified counts $\tilde{t}_{\cdot\mathbf{u}x}$ that appear in (2.16) arise from CRF representation of the HPYP by propagating the second constraint, $t_{\mathbf{u}s} = \mathbf{1}[c_{\mathbf{u}s} > 0]$ up through the hierarchy using (2.42). Starting at the leaves of the tree we have $c_{\mathbf{u}s} = c^x_{\mathbf{u}s}$; for each subsequent higher level, we set $t_{\mathbf{v}s} = \mathbf{1}[c_{\mathbf{v}s}]$ (where $\mathbf{v}$ is a child node of $\mathbf{u}$ and thus has already been computed), and compute $c_{\mathbf{u}s}$ according to (2.42). The $c_{\mathbf{u}s}$ computed in this way are exactly the modified counts $t_{\cdot\mathbf{u}x}$ of Kneser-Ney, i.e. they count the number of contexts $s\mathbf{u}$ with suffix $\mathbf{u}$ in which the symbol x appears.

As performing inference in the CRF representation of the HPYP amounts to computing the posterior distribution over the variables $\{t_{\mathbf{u}s}\}$, interpolated Kneser-Ney smoothing can be interpreted as an approximate inference scheme for this model which simply deterministically sets the values of $t_{\mathbf{u}s}$ as described above. This approximation often already yields good performance and thus can serve as a good starting point for other inference schemes.

2.4.3 Gibbs Sampling in the Chinese Restaurant Franchise

The basic building block for performing MCMC inference in HPYP-like models is a Gibbs sampler for the Chinese restaurant franchise. It can be seen as the hierarchical adaptation of the remove-add Gibbs sampler described in Algorithm 1 and has first been described in [Teh et al., 2006], adapted to the discrete language modelling case by [Teh, 2006a] and subsequently been used in almost all work on this type of model, including the Sequence Memoizer [Wood et al., 2009].

The main idea is to extend the CRP remove-add Gibbs sampler to the CRF in the same way the CRF is a hierarchical extension of the CRP: Customers are sequentially removed and then re-inserted into each restaurant; whenever a table becomes empty while removing a customer, the corresponding customer in the parent restaurant is removed. Conversely, whenever a customer sits at a new table, a corresponding customer is inserted into the parent restaurant (again potentially sitting at a new table, and leading to a customer being added to its parent restaurant, and so on recursively). As the conditional probabilities (2.3.6) for re-seating a customer of type s in restaurant $\mathbf{u}$ depend on the probability of s under the base distribution (given by (2.45)), which in turn depends on the seating arrangements in all restaurants from $\pi(\mathbf{u})$ to the root of the tree, these probabilities need to be re-computed whenever any of the parent restaurants' seating arrangements change during sampling.

One sweep of the Gibbs sampler consists of cycling through all restaurants $\mathbf{u}$ and performing a remove-add step for all customers in that restaurant. By visiting the restaurants in depth-first search order one can cache the probabilities under the base distribution along the path to the root, and thus avoid expensive unnecessary re-computations if the seating arrangements in the parent restaurants do not change.

As has been noted in [Teh, 2006a] storing the complete seating arrangements $A_{\mathbf{u}}$ (and implicitly a mapping of tables to customers in the parent restaurant) for each restaurant $\mathbf{u}$ requires an unnecessarily large amount of memory. One can dramatically reduce the amount of storage required for the seating arrangements by not storing the complete partition of the customers, but only the sufficient statistics, namely the number of customers sitting around each table. The state space of the sampler remains the state of partitions, but only the counts $c_{\mathbf{u}sk}, t_{\mathbf{u}s},\ k = 1,\ldots,t_{\mathbf{u}s}$ are represented. The main idea behind the Gibbs sampler for this representation proposed in [Teh, 2006a] is that given the sizes of the tables, all seating arrangements that satisfy these constraints are equiprobable (as (2.43) depends only on these sizes). We can thus imagine a sampler that interleaves the previously described remove/add sampler with a step that re-samples the seating arrangement conditioned on the sizes of the tables, i.e. which permutes the indices of customers sitting around tables. The resulting hypothetical sampler is equivalent the one summarized in Algorithm 2, which achieves the same effect without explicitly storing the seating arrangement, by re-instantiating just enough information required for the subsequent remove/add step, namely the table that the chosen customer sat at. The probability of a customer chosen uniformly at random (or, equivalently, a particular

customer after uniformly permuting their identities) sitting at a particular table k is proportional to the number of customers $c_{\mathbf{u}sk}$ siting at that table, so that the corresponding count $c_{\mathbf{u}sk}$ is decremented with probability

$$\mathrm{P}(\text{decrement } c_{\mathbf{u}sk}) = \frac{c_{\mathbf{u}sk}}{c_{\mathbf{u}s}} \tag{2.47}$$

and $t_{\mathbf{u}s}$ is decremented whenever a table becomes empty (i.e. $c_{\mathbf{u}sk} = 0$). Note that this is merely a conceptual description of the algorithm. An actual implementation can be made significantly faster by traversing the contexts $\mathbf{u}$ that occur in the training sequence in depth-first search order and avoiding unnecessary re-computations of $G_{\pi(\mathbf{u})}$, as well as interleaving DRAWPROPORTIONAL with the computation of the probabilities p_k.

MCMC for Hyperparameters

In addition to the latent variable of the CRF representation, the HPYP model has the sets of discount and concentration hyperparameters $\{d_{\mathbf{u}}\}$ and $\{\alpha_{\mathbf{u}}\}$ that need to be set or inferred alongside the latent variables. In the original work on the HPYP, Teh [2006a] compared two schemes: a Bayesian treatment placing priors on the hyperparameters and inferring their posterior distribution using an auxiliary variable MCMC sampler, and an optimization-based scheme which optimized log-loss with respect to the hyperparameters on a validation set given a fixed set of latent variables. We will discuss both options in the context of the Sequence Memoizer model in Section 5.7.2.

An alternative, simple and generally applicable strategy used in some subsequent work on HPYP-based models (e.g. [Wood and Teh, 2009]) is to use the Metropolis algorithm [Metropolis et al., 1953] (see e.g. [Gelman et al., 2004, Sec. 11.4] for an introduction) with a Gaussian proposal distribution and apply it to transformed, unconstrained parameters. In the HPYP model, the discount parameters are constrained to lie between zero and one, and the concentration parameters are constrained to be positive. We can parameterize the model in terms of unconstrained parameters $\tilde{d}_{\mathbf{u}}$ and $\tilde{\alpha}_{\mathbf{u}}$ e.g. by using a logistic sigmoid transformation for the discount parameters and an exponential transformation for the concentration parameters: $d_{\mathbf{u}} = (1 + e^{-\tilde{d}_{\mathbf{u}}})^{-1}$ and $\alpha_{\mathbf{u}} = \exp(\tilde{\alpha}_{\mathbf{u}})$. New parameter values $\theta^\star = (\{\tilde{\alpha}^\star_{\mathbf{u}}, \tilde{d}^\star_{\mathbf{u}}\})$ are then proposed by adding zero-mean Gaussian noise to the old parameters values θ^{old}, and accepted or rejected using the standard Metropolis acceptance ratio:

$$\mathrm{P}(\text{accept } \theta^\star) = \min\left(1, \frac{L(\theta^\star)}{L(\theta^{\text{old}})}\right) \tag{2.48}$$

where $L(\theta)$ is the likelihood function (for fixed latent variables) and the proposal distribution does not appear due to its symmetry. If the proposed new value is not accepted, the old value θ^{old} is kept.

Algorithm 2 HPYP Table Sizes Gibbs Sampler [Teh, 2006a, Table 2]

procedure REMOVECUSTOMER($\mathbf{u}, s$)
  $k^\star \leftarrow$ DRAWPROPORTIONAL($(c_{\mathbf{u}sk})_{k=1,\ldots,t_{\mathbf{u}s}}$)
  $c_{\mathbf{u}sk^\star} \leftarrow c_{\mathbf{u}sk^\star} - 1$ ▷ Remove customer
  if $c_{\mathbf{u}sk^\star} = 0$ **then**
    REMOVECUSTOMER($\pi(\mathbf{u}), s$)
  end if
end procedure

procedure ADDCUSTOMER($\mathbf{u}, s$)
  Add customer of type s to the $\mathbf{u}$ *restaurant*
  for $k = 1, \ldots, t_{\mathbf{u}s}$ **do**
    $p_k \leftarrow (c_{\mathbf{u}sk} - d)$
  end for
  $k^+ \leftarrow t_{\mathbf{u}s} + 1$
  $p_{k^+} \leftarrow (\alpha + d\, t_{\mathbf{u}\cdot}) G_{\pi(\mathbf{u})}(s)$
  $k^\star \leftarrow$ DRAWPROPORTIONAL($p_1, \ldots, p_{t_{\mathbf{u}s}}, p_{k^+}$)
  if $k^\star \neq k^+$ **then**
    $c_{\mathbf{u}sk} \leftarrow c_{\mathbf{u}sk} + 1$ ▷ Sit at an existing table
  else
    $c_{\mathbf{u}sk^+} = 1$ ▷ Create a new table
    ADDCUSTOMER($\pi(\mathbf{u}), s$)
  end if
end procedure

procedure REMOVEADDSAMPLER($\{\mathbf{u}, c_{\mathbf{u}sk}, t_{\mathbf{u}s}\}$)
  for $j = 1, \ldots,$ # sampling iterations **do**
    for $\mathbf{u} \in \mathscr{U}$ **do**
      for $s \in \Sigma, i = 1, \ldots, c_{\mathbf{u}s}$ **do**
        REMOVECUSTOMER($\mathbf{u}, s$)
        ADDCUSTOMER($\mathbf{u}, s$)
      end for
    end for
    After burn-in, save current counts as sample
  end for
end procedure

CHAPTER 3

Sequence Memoizer

The name *Sequence Memoizer* (SM) [Wood et al., 2011] has been coined for the model we originally proposed in [Wood et al., 2009], where it was referred to as *A stochastic memoizer for sequence data.* It can most easily be described at a high level as an extension of the HPYP language model described in Section 2.4 to unbounded context lengths. Longer context lengths are generally desirable as they allow long-range dependencies to be exploited, as long as the resulting data sparsity problem and computational burden can be managed. While it is conceptually unproblematic to make L large in the original HPYP model (e.g. by setting L to the length of the training data N, so that the context length is effectively unbounded), this approach is computationally problematic, as the number of nodes in the context tree and thus the number of random variables that need to be inferred grows as $O(N^2)$. Our main contribution in [Wood et al., 2009] is the insight that the number of nodes can be reduced to $O(N)$, thus making models with unbounded maximal context length computationally feasible.[1]

Based on the promising results in [Wood et al., 2009], we developed further extensions to make the model more scalable both in terms of space complexity (by using the compact representation described in Section 4.3) and in terms of time complexity (by using an online inference procedure described in Chapter 5), and improved the performance of the model by enlarging the range of allowed hyperparameters (Wood et al. [2009] had to fix $\alpha = 0$) and using different

[1] Note that while the original paper not only claimed to reduce the model size to a linear number of nodes, but also that the model "can be represented in time and space linear in the length of the training sequence" [Wood et al., 2009, Abstract], this was not strictly true in the original formulation for two reasons: The compact representation (see Section 4.3) had not been developed yet, meaning that for inference the quantities $c_{\mathbf{u}sk}$ needed to be represented in each node, requiring space growing linearly with *N for each node* in the worst case. Further, inference using the add/remove Gibbs sampler (Algorithm 2) scales as the number of customers in each node, which in the worst case is $O(N)$ for each node.

ways of fitting these (see Section 5.7.2). Other authors have also presented further developments of the Sequence Memoizer model in different directions, most notably the work by Bartlett et al. [2010] who present a way of turning the SM into a true streaming method, requiring only a fixed, pre-determined amount of space.

We will describe the model underlying the SM and the two main ingredients for making inference in the model practical: the compact context tree of distributions (Section 3.2) and the coagulation and fragmentation properties of the Pitman-Yor process (Section 3.3). We will then present the improvements that make the model practical and more accurate: the extended hyperparameter range (Section 5.7.1), the compact representation (Chapter 4), online model construction, as well as more efficient online and offline inference and hyperparameter learning. In Chapter 6 we describe how these advances can be combined to obtain a state-of-the-art lossless data compression algorithm.

3.1 Model

The generative model underlying the Sequence Memoizer is essentially the same as for the Hierarchical Pitman-Yor process language model (2.41): A sequence $x_{1:T} = x_1, x_2, \ldots, x_T \in \Sigma^*$ is modelled using a set of conditional distributions,

$$P(x_{1:T}) = \prod_{i=1}^{T} P(x_i|x_{1:i-1}) = \prod_{i=1}^{T} G_{x_{1:i-1}}(x_i), \tag{3.1}$$

where $G_{\mathbf{u}}(s)$ is the conditional probability of the symbol $s \in \Sigma$ occurring after a context $\mathbf{u} \in \Sigma^*$ (the sequence of symbols occurring before s). The parameters of the model consist of all the conditional distributions $\{G_{\mathbf{u}}\}_{\mathbf{u}\in\Sigma^*}$, and are given a hierarchical Pitman-Yor process (HPYP) prior,

$$\begin{aligned} G_\varepsilon &\sim \mathrm{PY}(\alpha_\varepsilon, d_0, H) \\ G_{\mathbf{u}}|G_{\sigma(\mathbf{u})} &\sim \mathrm{PY}(\alpha_{\mathbf{u}}, d_{|\mathbf{u}|}, G_{\sigma(\mathbf{u})}) \qquad \text{for } \mathbf{u} \in \Sigma^* \backslash \{\varepsilon\}, \end{aligned} \tag{3.2}$$

where ε is the empty sequence, $\sigma(\mathbf{u})$ is the sequence obtained by dropping the first symbol in $\mathbf{u}$, $d_{\mathbf{u}} \in [0,1]$ is a depth-dependent discount parameter, $\alpha_\varepsilon = \alpha_{\mathbf{u}} = 0$ are the PYP concentration parameters, and H is the overall base distribution over Σ (H is taken to be uniform over a finite Σ, as in the HPYP model). Comparing to the HPYP language model in (2.41), the main difference is the removal of K-th order Markov assumption in the likelihood and the corresponding finite-depth assumption on the hierarchical PYP prior on $G_{\mathbf{u}}$. Further, the concentration parameters $\alpha_{\mathbf{u}}$ are fixed to zero (in the original formulation

described in [Wood et al., 2009]; we will weaken this assumption in Section 5.7.1), in order to be able to analytically marginalize out some distributions using the coagulation-fragmentation property (Section 2.3.5).

As specified, the model has an unbounded number of hyperparameters $d_{|\mathbf{u}|}$ for $|\mathbf{u}| = 0, 1, 2, \ldots$. Wood et al. [2009] get around this by only using distinct discount parameters $d_{|\mathbf{u}|}$ for the first L levels, and using the same discount $d_{|\mathbf{u}|} = d_\infty$ for all $|\mathbf{u}| \geq L$. Wood et al. [2009] used $L = 4$ and initialized the discounts at $d_{[0,1,2,3,\infty]} = (.62, .69, .74, .80, .95)$. In later work [Gasthaus et al., 2010], and unless otherwise noted in this thesis, we set $L = 10$ and $d_{[0:9]} = (0.05, 0.7, 0.8, 0.82, 0.84, 0.88, 0.91, 0.92, 0.93, 0.94)$, $d_\infty = 0.95$.[2] An alternative to fixing the the discount parameters for depth greater than L to some constant value is to use a function that depends on $|\mathbf{u}|$, as done in [Bartlett and Wood, 2011], who for $|\mathbf{u}| \geq L$ choose $d_{|\mathbf{u}|} = d_\infty^{\beta^{|\mathbf{u}|+1-L}}$ for $\beta \in (0, 1)$, so that $d_{|\mathbf{u}|} \to 1$ as $|\mathbf{u}| \to \infty$.

Equation (3.2) defines a prior over an infinitely deep tree-structured hierarchy of random distributions. The first step in making inference in this model tractable is to notice that given a finite training sequence $x_{1:T}$ of length T, only the distributions that are indexed by a context that is a prefix of $x_{1:T}$ will be associated with a likelihood term, and these distributions only depend on their ancestors in the hierarchy (for each context $\mathbf{u}$, the ancestors correspond to the nodes indexed by contexts that are suffixes of $\mathbf{u}$). The resulting set of distributions that we need to infer is finite, but consists of all the distributions indexed by all substrings of $x_{1:T}$ which contains $O(T^2)$ contexts (we will denote this set by $\tilde{\mathscr{C}}(x_{1:T})$). The main insight in [Wood et al., 2009] was that a further reduction in the number of "interesting" distributions to a set of size $O(T)$ is possible: By making use of a coagulation-fragmentation property of the PYP we can analytically marginalize out all distributions that only have one child in $\tilde{\mathscr{C}}(x_{1:T})$ and are not a prefix of $x_{1:T}$. Wood et al. [2009] noted that the resulting tree of distributions has the form of a suffix tree, which can easily be seen to contain at most $2T$ nodes and which can be built from the input sequence in linear time.

[2] These values were originally chosen through manual tuning on the Brown corpus validation set, i.e. without extensive hyperparameter sweeps. However, they turn out to perform surprisingly well on many types of data, and often perform on par with or not much worse than the optimal values found either by optimization on a validation set or MCMC inference (see Section 5.7.2).

3.2 Context Trees

As can be seen from (3.1), given a finite observation sequence $x_{1:T}$ only a finite subset of all possible contexts $\mathbf{u} \in \Sigma^*$ are associated with a likelihood term. This set of "active" contexts corresponds to the set of prefixes $\text{Pre}(x_{1:T}) = \{x_{1:i} \mid i = 0, \ldots, T\}$ of $x_{1:T-1}$. According to the HPYP prior (3.2) each such context $\mathbf{u} \in \text{Pre}(x_{1:T-1})$ is given a PYP prior whose base distribution $G_{\sigma(\mathbf{u})}$ is associated with the longest proper suffix $\sigma(\mathbf{u})$ of $\mathbf{u}$, which in turn has a base distribution associated with $\sigma(\sigma(\mathbf{u}))$ and so on. The entire set contexts (and associated distributions) involved in the generation of the data is thus the union of all suffix sets for all prefixes of $x_{1:T-1}$ (equal to the set of all substrings of $x_{1:T-1}$), $\tilde{\mathscr{C}}(x_{1:T-1}) = \bigcup_{\mathbf{u} \in \text{Pre}(x_{1:T-1})} \text{Suf}(\mathbf{u})$, where $\text{Suf}(\mathbf{x}) = \{\mathbf{v} \mid \mathbf{uv} = \mathbf{x}; \mathbf{u}, \mathbf{v} \in \Sigma^*\}$ denotes the suffix set. When this set of all substrings is arranged in the same tree structure as the HPYP prior, i.e. with the parent of a string $\mathbf{u} = s\mathbf{v}$ given by its longest proper suffix $\sigma(\mathbf{u}) = \mathbf{v}$, we refer to this structure as a *context trie*.

The "trick" underlying the Sequence Memoizer is that it is possible to analytically marginalize out all distributions corresponding to contexts that only have one child in the context trie and don't have any likelihood terms associated with them (i.e. are not a prefix of $x_{1:T-1}$). The resulting *context tree*[3] only contains $O(T)$ nodes, T nodes corresponding to the prefixes and at most T other inner nodes. Figure 3.1 illustrates this relationship between the context trie and the context tree. More formally, the node set of a context trie $\tilde{\mathscr{T}}(x_{1:T})$ for the string $x_{1:T} \in \Sigma^*$ is the set $\tilde{\mathscr{C}}(x_{1:T})$ of all substrings of $x_{1:T}$ (including the empty string ε, which is the root), the edges are labeled with symbols $s \in \Sigma$, and there exists an edge $\mathbf{u} \xrightarrow{s} \mathbf{v}$ between nodes $\mathbf{u}$ and $\mathbf{v}$ iff $\mathbf{v} = s\mathbf{u}$. The leaf nodes correspond to the prefixes of $x_{1:T}$. Note that there is a one-to-one correspondence between the edge labels along the path from the root to any node $\mathbf{u}$ and the node label $\mathbf{u}$. The node set of a context tree $\mathscr{T}(x_{1:T})$ consists of all prefixes $\text{Pre}(x_{1:T})$ of $x_{1:T}$, as well as all nodes that have more than one child in $\tilde{\mathscr{T}}(x_{1:T})$, i.e. $\{\mathbf{u} \mid \exists s, t \in \Sigma : s \neq t \wedge s\mathbf{u} \in \tilde{\mathscr{C}}(x_{1:T}) \wedge t\mathbf{u} \in \tilde{\mathscr{C}}(x_{1:T})\}$. The edges in $\mathscr{T}(x_{1:T})$ are labeled with strings $\mathbf{w} \in \Sigma^+$, and there exists an edge $\mathbf{u} \xrightarrow{\mathbf{w}} \mathbf{v}$ between nodes $\mathbf{u}$ and $\mathbf{v}$ in $\mathscr{T}(x_{1:T})$, iff there exists a path $\mathbf{u} \xrightarrow{s_1} \mathbf{u}_1 \xrightarrow{s_2} \mathbf{u}_2 \xrightarrow{s_3} \ldots \xrightarrow{s_h} \mathbf{v}$ in the context trie with $\mathbf{w} = s_1 s_2 s_3 \cdots s_h$ and no other node $\mathbf{u}_1, \mathbf{u}_2, \ldots$ along the path is in the node set of $\mathscr{T}(x_{1:T})$. In other words, the nodes missing from the context tree

[3] This terminology, which follows the terminology used in the literature on suffix trees is somewhat unfortunate, as of course the *context trie* is also a tree. A different terminology that is sometimes used with suffix trees is to add the qualifier "compact" or "compressed" when referring to the context tree and the qualifier "atomic" when referring to the full trie [Giegerich and Kurtz, 1997], but this can lead to further confusion when used in conjunction with the (unrelated) *compact representation* (Section 4.3) or applications to data compression (Chapter 6).

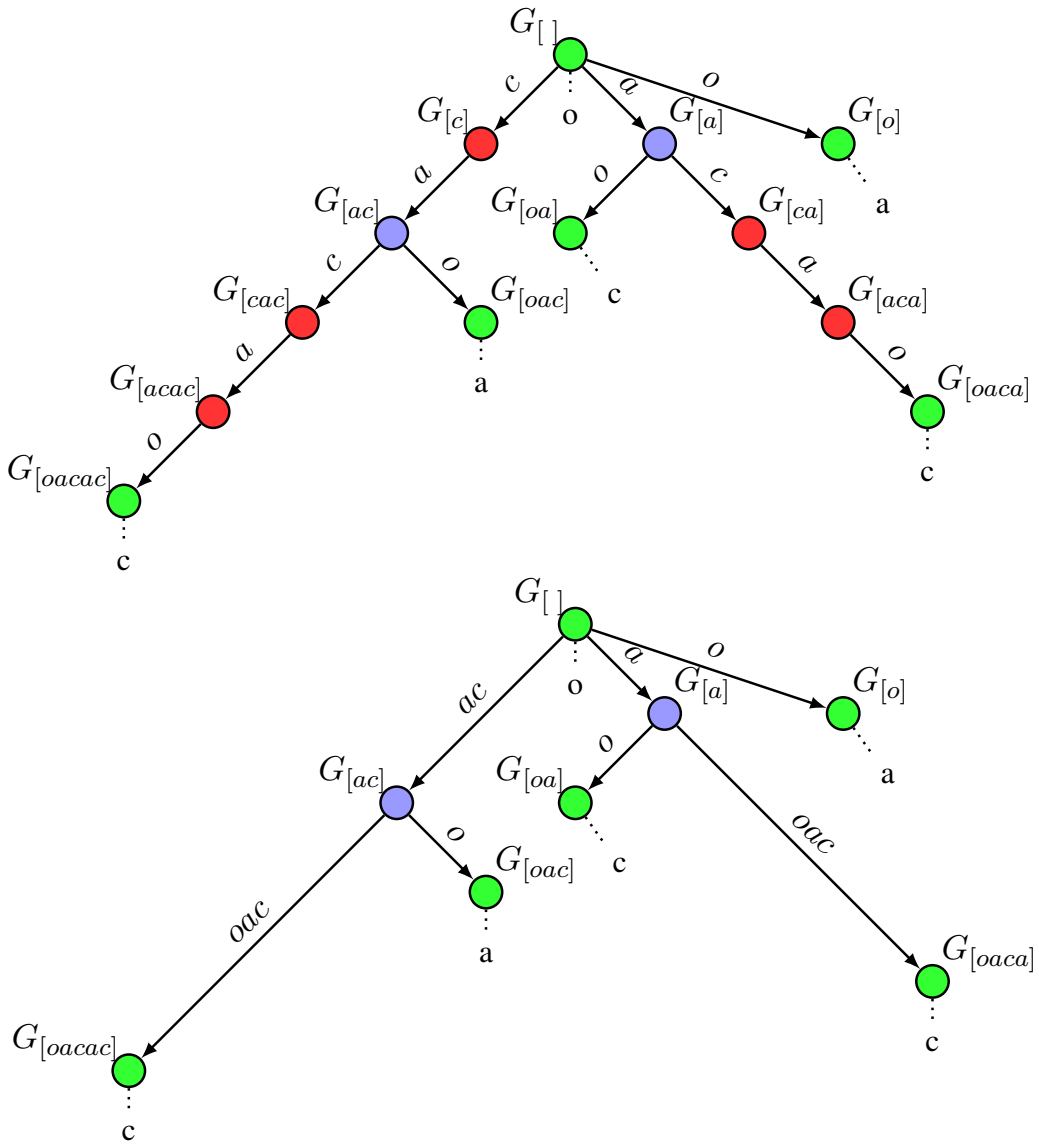


Figure 3.1: Context trie (top) and context tree (bottom) for the string `oacacc`. The nodes in trie correspond to all substrings of `oacac`. Shown in green are the nodes associated with likelihood terms, which correspond to the prefixes of `oacac`, namely ε, `o`, `oa`, `oac`, `oaca`, and `oacac`. Shown in red and blue are the parents of these nodes in the HPYP prior, where red nodes have the special property that they only have one child. In the context tree the red nodes are removed and replaced with edges containing multiple symbols. The trie has $O(T^2)$ nodes (e.g. consider the string $\mathtt{a}^T\mathtt{b}^T$), whereas the tree has $O(T)$ nodes (T nodes corresponding to prefixes and $\leq T$ other internal nodes).

(relative to the context trie) are precisely those that have exactly one child in the context trie (and are not a prefix of $x_{1:T}$), and the edge labels are simply the concatenation of the symbols along the corresponding path in the context trie. The context tree *implicitly* represents the substrings $\mathbf{u}$ that are nodes in the context trie but not in the context tree using edges labeled with strings. We will denote the (unique) parent of a context $\mathbf{u}$ in the context tree by $\pi(\mathbf{u})$ (i.e. there exists a $\mathbf{w} \in \Sigma^+$ such that $\pi(\mathbf{u}) \xrightarrow{\mathbf{w}} \mathbf{u}$ is an edge in the context tree), and the sequence of "implicit" contexts along the path from $\pi(\mathbf{u})$ to $\mathbf{u}$ (i.e. the nodes along the corresponding path in the context trie) by $(\pi(\mathbf{u}) \rightsquigarrow \mathbf{u})$ (excluding both $\pi(\mathbf{u})$ and $\mathbf{u}$).[4]

The main questions that remain to be addressed are how to construct the context tree (without constructing the trie first) and how to perform the analytic marginalization, and these will be answered in the following sections. An additional, more subtle issue that arises when making predictions on previously unseen data is how to make predictions with contexts that are not explicitly represented in the context tree. This will be elaborated on in Section 5.6.

3.2.1 Context Tree Construction

The problem of constructing the context tree is most readily addressed by reducing it to the well-studied problem of constructing *suffix trees*. Tree structures for storing all substrings of a string are a well-studied object in the area of string algorithms [Gusfield, 1997]. In particular, the well-known *suffix tree* data structure [Weiner, 1973; Ukkonen, 1995; Giegerich and Kurtz, 1997; Apostolico et al., 2016] is a building block for many fast exact and inexact string matching algorithms, as it allows substring search in time linear in the length of the substring and can be built in $O(T)$ for a length T string [Giegerich and Kurtz, 1997; Gusfield, 1997].[5] A suffix tree is a compact way of representing a *suffix trie*, a tree that has all substrings as nodes, all suffixes of the string as leaves, and where the parent of any node $\mathbf{u} = \mathbf{v}s$ is given by the longest proper prefix $\mathbf{v}$. Analogous to the context tree, a suffix tree retains only those nodes of the suffix trie that have more than one child (or are leaves). Like the context tree, it thus stores a subset of all substrings explicitly as nodes in the tree and all remaining substrings implicitly "in between" nodes. The main differences are that in a

[4] Note that the definition of $\pi(\mathbf{u})$ depends on the string $\mathbf{x}$ for which the context tree was constructed. We leave this dependence implicit as we are usually only considering a particular, fixed input string which will be clear from context.

[5] We will not describe the algorithms for linear time suffix tree construction here as they are fairly involved, but refer the reader to the detailed descriptions by Giegerich and Kurtz [1997] and Gusfield [1997]. Software libraries implementing these algorithms (in particular Ukkonen's algorithm [Ukkonen, 1995]) are also readily available.

suffix tree (a) the leaves correspond to suffixes of the string (as opposed to prefixes) and (b) the child-parent relationship of the underlying *suffix trie* is given by the longest proper prefix (as opposed to the longest proper suffix).

Perhaps unsurprisingly given this description, it can be seen that the context tree for a string corresponds exactly to the suffix tree for the reverse of the string if the node labels are read backwards. The suffix tree for the reverse of the string is also known as the *reverse prefix tree*, and plays a role in many suffix tree construction algorithms (see [Giegerich and Kurtz, 1997] for more details).

This is the connection exploited in [Wood et al., 2009]: The context tree can be constructed in linear time by applying Ukkonen's suffix tree construction algorithm [Ukkonen, 1995] to the reverse of the input string. While this allows the context tree to be built in linear time, there are drawbacks to this approach, namely that (a) the entire input sequence needs to be known ahead of time, making this approach unsuitable for applications where the model needs to be constructed incrementally, and (b) the algorithm is fairly complex and needs to be implemented carefully. It turns out that both of these issues can be addressed by a simple, "naïve" context tree construction algorithm which while having quadratic worst case time complexity is very fast in practice and is in fact (in a sense that we will make clear) optimal for the Sequence Memoizer use case.[6] This algorithm will be described in Section 5.1.

3.3 Marginalization

In order to reap the benefits of compactly representing the context tree, the inference procedure employed needs to be based on only those contexts which are explicitly represented. This is made possible by analytic marginalization of the non-represented distributions using the coagulation properties of the Pitman- Yor process (see Section 2.3.5). The original description of the SM in [Wood et al., 2009] relied on a special case of this result obtained by setting all concentration parameters $\alpha_{|\mathbf{u}|}$ to zero:

Theorem 3 (Simplified PYP Coagulation; adapted from [Pitman, 1999])**.** If $G_1 \sim \mathrm{PY}(0, d_1, H)$ and $G_2 \mid G_1 \sim \mathrm{PY}(0, d_2, G_1)$, then marginally $G_2 \sim \mathrm{PY}(0, d_1 d_2, H)$ for $0 < d_1 < 1$, $0 < d_2 < 1$.

[6] We note in passing that $O(n)$ online algorithms for reverse prefix tree construction *do* exist (e.g. Weiner's "repetition finder" [Weiner, 1973; Giegerich and Kurtz, 1997] which constructs the suffix tree starting from the end of the string), but are not necessarily more efficient for the task at hand for typical inputs as the constant factors involved are larger, and usually do require additional memory for storing additional pointers.

Recursively applying this marginalization to all distributions corresponding to the non-branching internal nodes of the context trie yields a description of the marginal distributions corresponding to the nodes in the context tree, where the discount parameters for the remaining distributions are simply the product of the discount parameters associated with all "implicit" distributions along the path to the parent node in the context tree. More precisely, for each context $\mathbf{u} \in \mathcal{T}$ we have the marginal prior:

$$G_{\mathbf{u}} | G_{\pi(\mathbf{u})} \sim \mathrm{PY}(0, \tilde{d}_{\mathbf{u}}, G_{\pi(\mathbf{u})}) \tag{3.3}$$

where we have defined the modified discount parameter

$$\tilde{d}_{\mathbf{u}} = d_{\mathbf{u}} \prod_{v \in (\pi(\mathbf{u}) \rightsquigarrow \mathbf{u})} d_{\mathbf{v}}. \tag{3.4}$$

3.4 Inference & Re-Instantiation

Inference in the Sequence Memoizer model is essentially identical to inference in the HPYP language model (2.4.3), but applied only to the distributions $G_{\mathbf{u}}$ corresponding to nodes in context tree and using the modified discounts (3.4). Any inference procedure developed for the HPYP model can thus be used for the SM (and vice versa), and Wood et al. [2009] employed the same Add/Remove Gibbs sampler as Teh [2006a] (Algorithm 2). Making predictions using the SM is also analogous to the HPYP model: the predictive probability (2.45) is averaged wrt. to the samples obtained from the sampler.

One complication arises when predictions are to be made for contexts $\mathbf{v} \in \tilde{\mathcal{T}}$ that are not explicitly represented in the context tree ($\mathbf{v} \notin \mathcal{T}$), and thus their associated distributions $G_{\mathbf{v}}$ and restaurants are not represented in the Chinese restaurant franchise. Wood et al. [2009] address this by proposing to "re-instantiate" the CRP representation of $G_{\mathbf{v}}$ from the representation of $G_{\mathbf{w}}$, where $\mathbf{v} \in (\pi(\mathbf{w}) \rightsquigarrow \mathbf{w})$ and $\mathbf{w}$ is a node in the context tree. Using the coagulation/fragmentation duality, a sample from the CRP representation of $G_{\mathbf{v}}$ and $G_{\mathbf{w}} \mid G_{\mathbf{v}}$ can be obtained from the representation of $G_{\mathbf{w}}$ using Theorem 2: Each table in the $G_{\mathbf{w}}$ restaurant is independently split according to a CRP with concentration parameter $-\tilde{d}_{\mathbf{w}}$ and discount parameter $d_{\mathbf{w}} \prod_{\mathbf{u} \in (\mathbf{v} \rightsquigarrow \mathbf{w})} d_{\mathbf{u}}$. The union of the resulting tables forms the sampled representation of $G_{\mathbf{w}} \mid G_{\mathbf{v}}$, and each resulting table contributes a customer to the representation of $G_{\mathbf{v}} \mid G_{\pi(\mathbf{w})}$ which are grouped in tables according to which table in the representation of $G_{\mathbf{w}} \mid G_{\pi(\mathbf{w})}$ they originated from. This is illustrated in Figure 3.2 (cf. also Figure 2.9). Note that the resulting seating arrangement satisfies the constraints (2.42) by construction.

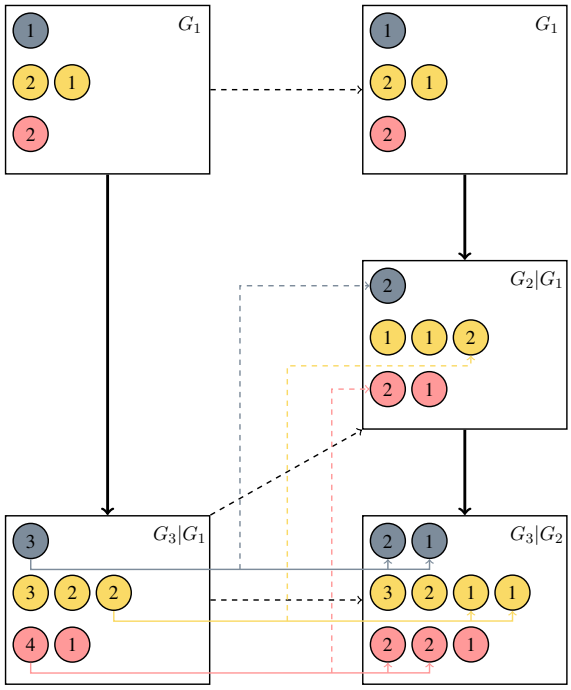


Figure 3.2: Illustration of the re-instantiation of the CRP representation of $G_2 \mid G_1$ that had been marginalized out. The circles represent tables in their label the number of customers sitting at them; the colors represent different sections. In order to re-instantiate $G_2 \mid G_1$, the customers at every table in the $G_3 \mid G_2$ restaurant are partitioned using a CRP, and the resulting tables form the customers in the re-instantiated $G_2 \mid G_1$ restaurant. They are grouped into tables according to the tables in $G_3 \mid G_1$ they originated from, so that $G_2 \mid G_1$ and $G_3 \mid G_1$ have the same number of tables, and are both consistent with G_1.

3.5 Sequence Memoizer vs. HPYP

The Sequence Memoizer model is an extension of the HPYP model to unbounded context lengths. One of the key properties of this model demonstrated in the original paper [Wood et al., 2009] is that it compares favorably to HPYP models with fixed context length in terms of predictive performance *and* computational complexity as the context lengths become larger. By using the context tree instead of the context trie data structure, the number of nodes/-contexts that are represented explicitly (determining the computational and memory complexity) in the SM model can be less than the number of nodes represented in the HPYP model with fixed context length. This is illustrated in Figure 3.3, which shows how the test set perplexity of the HPYP model decreases with increasing context length, approaching the performance of the SM, while the number of nodes in the context trie exceeds the number of nodes in the context tree for context lengths ≥ 4.

Figure 3.4 shows how the number of nodes grows as a function of input

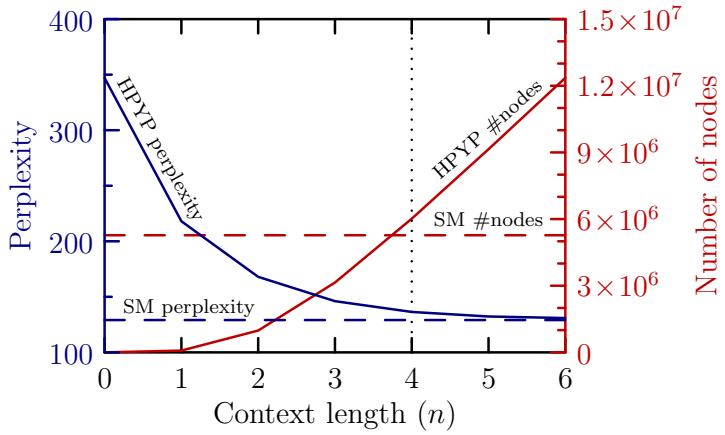


Figure 3.3: Shown in blue is the predictive performance of the SM model (dashed line) versus HPYP models with fixed context length n (solid line) as n varies (perplexity on the test set, lower is better). Shown in red is the number of nodes/contexts in the context tree of the SM model (dashed line) or the trie with depth n of the HPYP model (solid line). These results were obtained on the four million word NYT corpus, but are illustrative of the general behavior of these models on text data. This figure illustrates the general appeal of the SM model: In this case, for context lengths ≥ 4, the computational and memory complexity (which are both proportional to the number of nodes) of the trie-based HPYP model exceeds that of the SM model, while the SM model has better modelling performance. This suggests that the SM model is generally to be preferred over the HPYP for large enough context lengths.

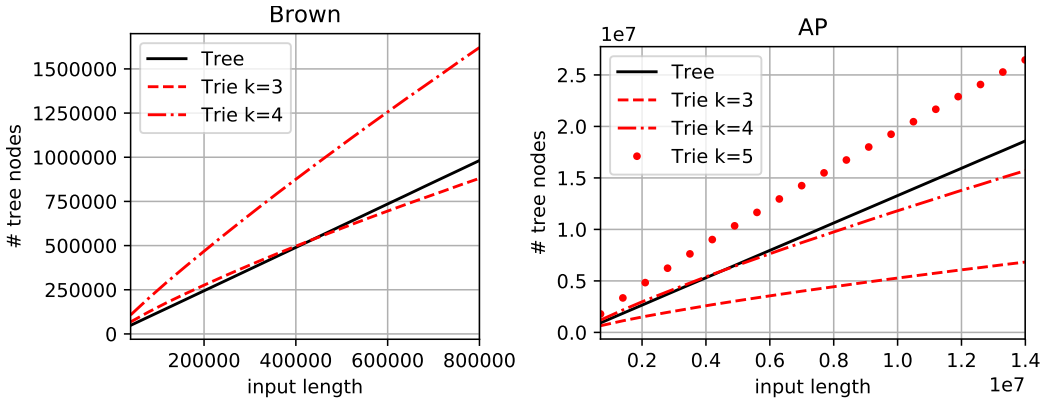


Figure 3.4: Number of tree nodes as a function of input size for the Brown corpus (left) and the AP News corpus (right). The number of nodes in the context tree used by the SM model is shown in black, and the number of nodes in tries are shown in red, with different lines showing different context lengths. For some data-dependent context size (4 for the Brown corpus, 5 for AP News) the number of nodes in the trie exceeds the number of nodes in the tree. Note that in practice the number of nodes in the tree significantly smaller than the $2N$ worst case.

Model	Perplexity
[Mnih and Hinton, 2009]	112.1
[Bengio et al., 2003]	109.0
4-gram Modified Kneser-Ney [Teh, 2006b]	102.4
4-gram HPYP [Teh, 2006b]	101.9
Sequence Memoizer [Wood et al., 2009]	96.9

Table 3.1: Perplexity of the AP news test set under the SM model compared against the HPYP, Modified Kneser-Ney, and two neural language models (as reported in [Wood et al., 2009]).

size for the context tree, and context tries with different maximum context lengths. For the Brown corpus data set, the number of nodes in the context tree is similar to that of a trie with depth 3, and a trie with depth 4 is significantly larger for all input sizes. On the AP news data set, the number of nodes in the context tree is slightly larger than that of a depth-4 trie, but smaller than that of trie with depth 5. Hence in both cases there exists a context length beyond which the context tree used by the SM model contains less nodes than a fixed-depth context trie.

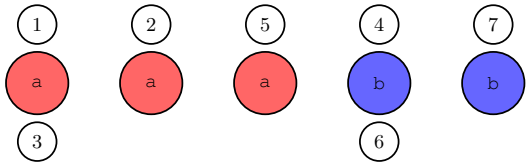
Table 3.1 (taken from [Wood et al., 2009]) shows the predictive performance of the SM model in terms of test set perplexity compared against the fixed-depth HPYP model, a variant of Kneser-Ney smoothing, as well as two neural language models.

CHAPTER 4

Sequence Memoizer: Representations

In order to perform inference in the Chinese Restaurant Franchise (CRF) representation of HPYP models, the main object that needs to be represented are the seating arrangements of customers around tables, as well as the relationships between tables in the child restaurant and the corresponding customers in the parent restaurant. The representation chosen crucially affects both the amount of memory required to represent the state of the sampler (and samples from the posterior) and the complexity and speed of the associated Monte Carlo inference procedures.

In the following we will describe four representations for the CRF (illustrated in Figure 4.1): the original CRF representation using partitions of [Teh et al., 2006], the counts/table-sizes representation developed in [Teh, 2006a] and used



Name	Representation	Size
PARTITIONS [Teh et al., 2006]	$\{\{1,3\},\{2\},\{5\},\{4,6\},\{7\}\}$	$O(c)$
COUNTS [Teh, 2006a]	$[2,1,1]$ $[2,1]$	$O(t)$
COMPACT [Gasthaus and Teh, 2010]	$(4,3)$ $(3,2)$	$O(1)$
HISTOGRAM [Blunsom et al., 2009]	$[1:2,2:1]$ $[1:1,2:1]$	$O(t)$

Figure 4.1: Illustration of the different representations in a restaurant with two different types (represented by colour).

in [Wood et al., 2009], the histogram representation developed by [Blunsom et al., 2009], and the compact representation developed in [Gasthaus and Teh, 2010] and used in [Gasthaus et al., 2010; Bartlett et al., 2010; Bartlett and Wood, 2011]. For completeness we also describe the level indicator representation of Chen et al. [2011] in Section 4.4, though we will not consider it in the sequel.

Each representation has different memory/speed trade-offs when combined with an appropriate inference method. The basic add/remove Gibbs sampler (Algorithm 2) operates by removing and re-adding customers according to the conditional posterior probabilities under a CRP partition (Eqns. (2.24a) and (2.47)), so that computing these probabilities and adding/removing customers and tables should be fast, while the direct sampler (Section 4.5) needs to evaluate (4.4) and set the counts $t_{\mathbf{u}s}$ directly.

4.1 Partition & Table Sizes Representation

As the Chinese restaurant franchise (2.43) describes a joint distributions over the partitions $\{A_{\mathbf{u}}\}$, the natural starting point is a representation where these partitions are represented explicitly, by associating each table with the customers sitting around it (or, equivalently, associating each customer with the table she sits at). This representation is equivalent to the one proposed for the two-level hierarchy in the original HDP paper [Teh et al., 2006], where (using the notation used in that paper) the variables t_{ji} represent the bottom level partition by storing the associated table index for each customer i in restaurant j and the variables k_{jt} represent the top level partition by storing the parent table index for each table t in restaurant j (i.e. the mapping from customers to tables in the parent restaurant).

As the table indices in the child restaurant correspond the customer indices in the parent restaurant, this representation explicitly maintains "pointers" from each table to its parent table, and so the dishes served at each table can be reconstructed deterministically by propagating them downwards from the top of the hierarchy. Gibbs sampling in this representation is simple, as the customers at each level are exchangeable and a simple remove-add sampler can be employed (see Section 2.4.3). In order to apply this representation to the case of multiple child restaurants, one additionally requires a way of associating customers coming from lower-level restaurants to the restaurant they came from, either by maintaining an explicit mapping or by ordering the child restaurants and ordering the customers accordingly.

Teh [2006a] noted that the full CRF representation described above is rather

wasteful and unnecessary in some cases: For language modelling, where the likelihood is a delta function, there is (a) no uncertainty about the identity of the dish served at each table and (b) no relevance to discovering the partition structure. Relevant for the likelihood of the data and for predictions made using the model is only the *number* of tables of each word type.[1] Based on this observation Teh [2006a] devised a sampler (reviewed in Section 2.4.3) that only requires storing the number of customers $c_{\mathbf{u}sk}$ sitting around *each* table $k = 1,\ldots,t_{\mathbf{u}s}$. We refer to this representation as the COUNTS or table-sizes representation.[2] As, in the worst case, there are as many tables as there customers (with each customer sitting by himself), the storage complexity of this representation grows as the number of customers. In addition to the bad worst-case complexity, this representation also performs poorly in practice. This is on the hand due to the fact that the representation needs to change size dynamically during inference, and on the other hand due to the high posterior probability for larger numbers of tables when the discount parameter is large (see Figures A.4 and A.5).

4.2 Histogram Representation

Blunsom et al. [2009] proposed an improvement to the table sizes representation that – while having the same worst-case space complexity – is much more efficient (in terms of both space and speed) in practice: Instead of storing the sizes $c_{\mathbf{u}sk}$ of all tables, the idea is to store a histogram of table sizes for each type. More formally, instead of storing per-table counts $c_{\mathbf{u}sk}$, one stores the counts $m_{\mathbf{u}sl} = \sum_k \mathbf{1}[c_{\mathbf{u}sk} = l], l = 1,\ldots,c_{\mathbf{u}s}$ using a sparse representation (i.e. only the values $m_{\mathbf{u}sl} > 0$).[3] The main advantage of this representation is that it is very well matched to the to power-law behavior of the PYP, in that it allows a large number of small tables to be represented efficiently. Further, it allows the conditional probabilities required for the add/remove sampler (Algorithm 2) to be computed efficiently: as the probability for a customer

[1] Given the number of tables for each word type in each restaurant in the hierarchy as well as the observed data, one can deterministically reconstruct the number of customers of each type in each restaurant. However, for computational efficiency, is is advisable to also store the number of customers of each type explicitly.

[2] For computational reasons it is advisable to additionally store the number of customers $c_{\mathbf{u}s}$ of each type as well as the total number customers $c_{\mathbf{u}\cdot}$ and tables $t_{\mathbf{u}\cdot}$ (all of which are needed to evaluate (2.24a) and (2.31)), even though these quantities can be deterministically reconstructed given the table sizes $c_{\mathbf{u}sk}$.

[3] In practice, this can be implemented e.g. using a hash map or a tree-based map. We found an implementation based on two dynamically resized arrays, one of which contains the sorted non-zero keys l very efficient, as it allows keys to be incremented very efficiently, which is one of the operations required by the add/remove sampler.

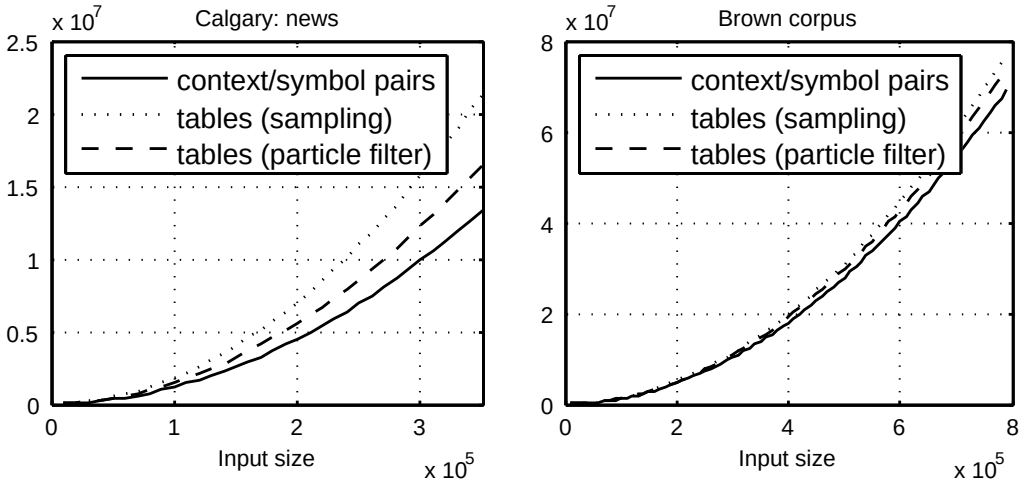


Figure 4.2: Number of context/symbol pairs and total number of tables (counted after particle filter initialization and 10 sampling iterations using the compact original sampler) as a function of input size. The left panel shows the counts obtained from a byte-level model of the `news` file in the Calgary corpus, whereas the right panel shows the counts for a word-level model of the Brown corpus (training set). The space required for the compact representation is proportional to the number of context/symbol pairs, whereas for the full representation it is proportional to the number of tables. In practice, the difference in amount of memory required is even more pronounced, as the constant-sized compact representation can be stored more efficiently with less overhead and memory fragmentation. For example, our implementation `libPlump` (Section 6.6) has a peak memory consumption of ≈ 650 MB using the compact representation and ≈ 950 MB when using the COUNTS representation, when performing particle filter inference on `news`. Note also that sampling tends to increase the number of tables over the particle filter initialization in this setting.

joining a particular table only depends on the size of the table, all tables of the same size can be considered together, so that the conditional probability of joining any of the $m_{\mathbf{u}sl}$ tables of size l is simply proportional to $(l-d) \times m_{\mathbf{u}sl}$. Similarly, during removal we need to uniformly pick a customer to remove, so that the probability of removing a customer from a table of size l is $\propto l \times m_{\mathbf{u}sl}$. Traversal of the histogram, computation of the probabilities and sampling can be interleaved to further improve efficiency (see Algorithms 1/2 in [Blunsom et al., 2009] which can trivially be extended to the PYP case).

4.3 Compact Representation

Recall that the predictive distribution (2.31) only depends on the total number of customers $c_{\mathbf{u}s}$ and tables $t_{\mathbf{u}s}$ for each symbol, as well as their sums $c_{\mathbf{u}\cdot}$ and $t_{\mathbf{u}\cdot}$. The idea behind what we call the *compact representation* introduced in [Gasthaus and Teh, 2010] is to only store these minimally necessary statistics, but no information about the sizes of the individual tables nor about the seating arrangements. The storage complexity of this representation is

constant wrt. the number of observations and in implementations does not require any overhead due to dynamically resizing data structures. As for the histogram representation, the ADDCUSTOMER operation in the add/remove sampler (Algorithm 2) can be performed efficiently, as we only need to compute the probability of a customer either sitting at any one of the existing tables, or starting a new table (in which case $t_{\mathbf{u}s}$ is incremented). The probability of joining any of the existing tables or a new table are:

$$\text{P(join existing table)} \propto c_{\mathbf{u}s} - t_{\mathbf{u}s} d_{\mathbf{u}} \tag{4.1a}$$

$$\text{P(start new table)} \propto \alpha_{\mathbf{u}} + d_{\mathbf{u}} t_{\mathbf{u}\cdot} G_{\pi(\mathbf{u})}(s) \tag{4.1b}$$

In order to perform the REMOVECUSTOMER operation, we need to compute the probability that a randomly chosen customer sits at a table by himself, so that after removing him from the restaurant, $t_{\mathbf{u}s}$ would need to be decremented. This turns out to be slightly more complicated: The probability that a randomly chosen customer sits by himself is equal to the total probability mass of $\mathscr{A}_{c_{\mathbf{u}s}, t_{\mathbf{u}s}}$ partitions where $c_{\mathbf{u}s} - 1$ customers sit around $t_{\mathbf{u}s} - 1$ tables. Using (2.26) and summing over all partitions where $c_{\mathbf{u}s} - 1$ customers sit around $t_{\mathbf{u}s} - 1$ tables using (2.27) we obtain

$$\text{P(decrement } t_{\mathbf{u}s}) = \frac{S_d(c_{\mathbf{u}s} - 1, t_{\mathbf{u}s} - 1)}{S_d(c_{\mathbf{u}s}, t_{\mathbf{u}s})}. \tag{4.2}$$

Due to Stirling numbers involved in this expression its computation requires $O(c^2_{\mathbf{u}s})$ operations.[4] When using the add/remove sampler this representation thus trades off storage complexity for computational complexity. However, when combined with sequential Monte Carlo inference (Chapter 5), this representation is computationally more efficient, as only a single probability (4.1b) needs to be evaluated for each restaurant visited on the path. The compact representation can also be combined with the direct Gibbs sampler (Section 4.5) and "fractional counts" approximate inference scheme (Section 5.4). The space savings that are possible using the compact representation are shown for the Brown corpus and the `news` file of the Calgary corpus in Figure 4.2.

4.4 Level Indicator Representation

For completeness, we also describe the *table indicator* representation developed in [Chen et al., 2011; Buntine and Hutter, 2012], even though we will not make use of it. In this representation of the CRP, each customer is associated

[4] When there are only a few distinct discount parameters (e.g. in the HPYP case), the Stirling numbers can be pre-tabulated and re-used, making this sampler computationally competitive.

with a binary indicator variable that determines whether that customer is "responsible" for creating a new table. If we denote the indicator associated with x_i by r_i, we can compute the multiplicities t_s by summing r_i over all i such that $x_i = s$. As there are $\binom{c_s}{t_s}$ ways of setting the indicators to obtain a count t_s, the joint probability of the observations and the indicators r_i is given by [Buntine and Hutter, 2012, Eq. (11)]

$$\mathrm{P}(x_{1:N}, r_{1:N}) = f^{\mathrm{CT}}_{a,d}(\mathbf{c}, \mathbf{t}) \prod_{s \in \Sigma} \frac{H(s)^{t_s}}{\binom{c_s}{t_s}}. \tag{4.3}$$

where the counts $\mathbf{c}, \mathbf{t}$ are obtained from $x_{1:N}, r_{1:N}$ and f^{CT} is defined in (2.34). Building on this, [Chen et al., 2011] further develop a representation for the hierarchical setting, which associates each observation at the leaf nodes with an indicator variable that denotes up to which level in the hierarchy this data item has created a new table. The main difficulty for performing Gibbs sampling in this representation is that the settings of the indicators for other data items constrain the allowed values for the indicator being sampled: if another data item is sharing a table created by the data item under consideration at some level in the hierarchy, then the level indicator cannot be changed beyond that point. Further, as in the compact representation, the conditional distribution for the level indicators involves a ratio of Stirling numbers and thus has the same computational drawbacks in hierarchies with many distinct discount parameters where the Stirling numbers cannot be pre-tabulated.

4.5 Gibbs Sampling in the Compact Representation

While the add/remove Gibbs sampler has been used in almost all work on the HDP/HPYP/SM, it is of course not the only possible MCMC scheme for this model.[5] In particular, in the delta function likelihood setting where the partitions $A_{\mathbf{u}}$ are not meaningful and not required for making predictions, samplers operating directly in other representations such as the compact or histogram representation described above can be designed.

In [Gasthaus and Teh, 2010] we described a Gibbs sampler for the compact representation which samples the number of tables $t_{\mathbf{u}s}$ of a particular type s directly from its conditional distribution given the settings of all other variables (we call this the DIRECT sampler). Like the add/remove sampler, it cycles through all contexts $\mathbf{u}$ and symbols s, sampling a new value for $t_{\mathbf{u}s}$ for each

[5] In the following we will not distinguish further between the HPYP and SM model, as in terms of inference they are identical. We will generically refer to the parent context as $\pi(\mathbf{u})$, and the parameters associated with a context $\mathbf{u}$ as $\alpha_{\mathbf{u}}$ and $d_{\mathbf{u}}$.

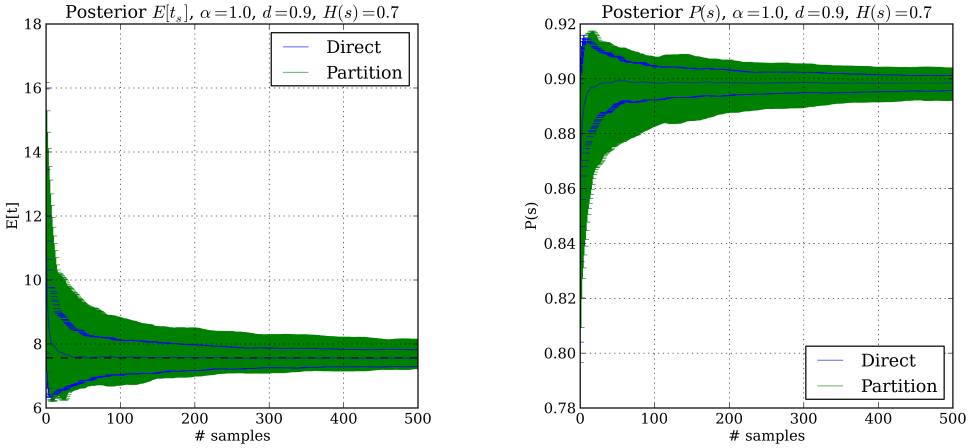


Figure 4.3: Comparison of the DIRECT sampler and the ADDREMOVE sampler on a simple two-level hierarchical model. Twenty customers of the same type are inserted into the bottom restaurant using the particle filter initialization. Each sampler then alternates between the bottom and the top restaurant. The ADDREMOVE sampler removes and re-inserts all customers in one iteration, whereas the DIRECT sampler directly draws one sample from the conditional distribution of $t_{\mathbf{u}s}$. Shown are the mean and standard deviation (resulting from 100 repeated runs of the sampler) of the resulting expected number of tables in the top restaurant (left) and the posterior predictive probability of observing the same type again (right). As expected, the initialization has a bias to overestimate the number of tables, and the DIRECT sampler mixes more quickly and is thus able to converge on the correct value (dashed line) faster.

context-symbol pair $\mathbf{u}s$ in turn. Since each $c_{\mathbf{u}s}$ is determined by $c^x_{\mathbf{u}s}$ and the $t_{\mathbf{v}s}$ at child restaurants $\mathbf{v}$ through (2.42), it is sufficient to update each $t_{\mathbf{u}s}$, which for $t_{\mathbf{u}s}$ in the range $\{1,\ldots,c_{\mathbf{u}s}\}$ has conditional distribution (obtained by dropping all terms from the joint distribution (2.44) that do not involve $t_{\mathbf{u}s}$):

$$P(t_{\mathbf{u}s}|\text{rest}) \propto \frac{[\alpha_{\mathbf{u}}+d_{\mathbf{u}}]_{d_{\mathbf{u}}}^{t_{\mathbf{u}\cdot}-1}}{[\alpha_{\pi(\mathbf{u})}+1]_1^{c_{\pi(\mathbf{u})\cdot}-1}} S_{d_{\mathbf{u}}}(c_{\mathbf{u}s},t_{\mathbf{u}s}) S_{d_{\pi(\mathbf{u})}}(c_{\pi(\mathbf{u})s},t_{\pi(\mathbf{u})s}), \tag{4.4}$$

where $t_{\mathbf{u}\cdot}$, $c_{\pi(\mathbf{u})\cdot}$ and $c_{\pi(\mathbf{u})s}$ all depend on $t_{\mathbf{u}s}$ through the constraints (2.42).[6]

We compared the DIRECT and the re-instantiating ADDREMOVE samplers both on a simple two-level model with synthetic data, as well as on the Sequence Memoizer model using the Brown corpus data. The results are shown in figures 4.3 and 4.4. On the synthetic data set, the DIRECT sampler mixes

[6] One caveat with this sampler is that we need to compute $S_{d_{\mathbf{u}}}(c,t)$ for all $1 \le c,t \le c_{\mathbf{u}s}$. If $d_{\mathbf{u}}$ is fixed these can be precomputed and stored, trading off memory for computational efficiency. However, if many restaurants have distinct discount parameters (as in the SM setting) storing these pre-tabulated Stirling numbers might be infeasible. If $d_{\mathbf{u}}$ is updated during sampling, the Stirling numbers will need to be re-computed each time as well, at a cost of $O(c^2_{\mathbf{u}s})$ per iteration. Further, $S_d(c,t)$ typically has very high dynamic range, so care has to be taken to avoid numerical under-/overflow (e.g. by performing the computations in the log domain, involving many expensive log and exp computations). Devising a way to efficiently compute or approximate these Stirling numbers is an interesting avenue for further research (see also Section 8.4 in [Buntine and Hutter, 2012].

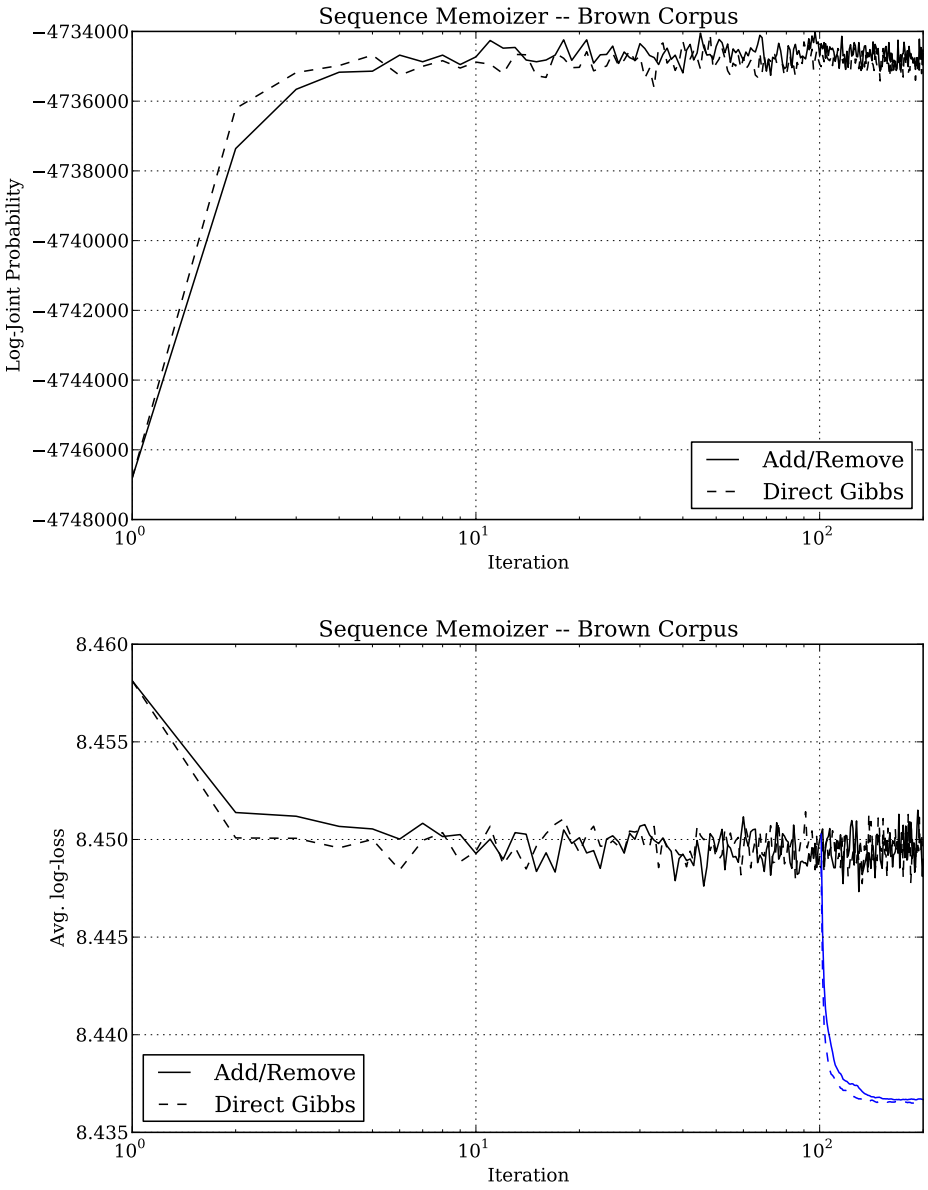


Figure 4.4: Comparison of the DIRECT sampler and the ADDREMOVE sampler for the Sequence Memoizer model on the Brown corpus. The top figure show the log-joint probability of the current sample while the bottom figure shows the average (negative) log predictive probability on the test set. Starting from the particle filter initialization, both samplers converge after only a few iterations, with the direct sampler being slightly faster as expected. The predictive probability improves only slightly (≈ 0.01 bits) through sampling. Averaging predictions from multiple samples after burn-in (shown in blue in the bottom figure) yields another improvement of approximately the same size, and only very few samples (≈ 10) are needed to achieve most of the improvement. Time per iteration was $43s$ for the direct sampler and $26s$ for the add/remove sampler in the compact representation. For comparison, the re-instantiating add/remove sampler takes $4s$ per iteration, and the resulting average log-loss is 8.44 bits for all samplers.

more quickly and is more efficient in estimating the correct expected number of tables and predictive probability. In the SM model (Figure 4.4) the differences are not as pronounced.

A viable alternative to this Gibbs sampling approach that we did not explore further could be a Metropolis-Hastings sampler, where a new setting of $t_{\mathbf{u}s}$ (either for each context-symbol pair $\mathbf{u}s$ separately or jointly for all $\{\{t_{\mathbf{u}s}\}_{s\in\Sigma}\}$) is proposed using some proposal distribution (e.g. uniform on $\{1,\ldots,c_{\mathbf{u}s}\}$, by incrementing/decrementing the current value, or by using a particle filter) and accepted or rejected using the standard MH accept/reject probability (2.48) with (2.44) as target. As computing the Stirling numbers $S_d(c_{\mathbf{u}s}, t_{\mathbf{u}s})$ does involve computing them for all smaller values of $c_{\mathbf{u}s}$ and $t_{\mathbf{u}s}$ (see Section A.2), evaluating (2.44) in (2.48) is however not necessarily much cheaper than computing (4.4) for all possible values of $t_{\mathbf{u}s}$.

Another strategy for sampling in the compact representation that avoids computing the Stirling numbers is to re-instantiate the seating arrangement by sampling $A_{\mathbf{u}} \sim \mathrm{CRP}_{c_{\mathbf{u}s}t_{\mathbf{u}s}}(d_{\mathbf{u}})$ from its conditional distribution given $c_{\mathbf{u}s}, t_{\mathbf{u}s}$ (see below), then performing normal add/remove Gibbs sampling. This produces a new number of tables $t_{\mathbf{u}s}$ and the seating arrangement can be discarded. Note however that when $t_{\mathbf{u}s}$ changes this sampler will introduce changes to ancestor restaurants (by adding or removing customers), so these will need to have their seating arrangements instantiated as well. To implement this sampler efficiently, we visit restaurants in depth-first order, keeping in memory only the seating arrangements of all restaurants on the path to the current one. The computational cost is $O(c_{\mathbf{u}s}t_{\mathbf{u}s})$, but with a potentially smaller hidden constant (no log/exp computations are required). In our implementation, using this re-instantiation strategy proved to be much more efficient for the SM model than either of the samplers that required explicit computation of the Stirling numbers, even when those were cached locally during sampling. A comparison in terms of time per iteration is shown in Figure 4.5.

4.6 Sampling from $\mathbf{CRP}_{ct}$

For the re-instantiating Gibbs sampler and the fragmentation operation in the compact representation we need to be able to sample a partition $A \sim \mathrm{CRP}_{ct}(d)$ of $[c]$ with a fixed number of blocks t according to (2.26), repeated here for convenience:

$$\mathrm{CRP}_{ct}(A|d) = \frac{\prod_{a\in A}[1-d]_1^{|a|-1}}{S_d(c,t)} \quad \text{for each } A \in \mathscr{A}^t_{[c]}. \tag{4.5}$$

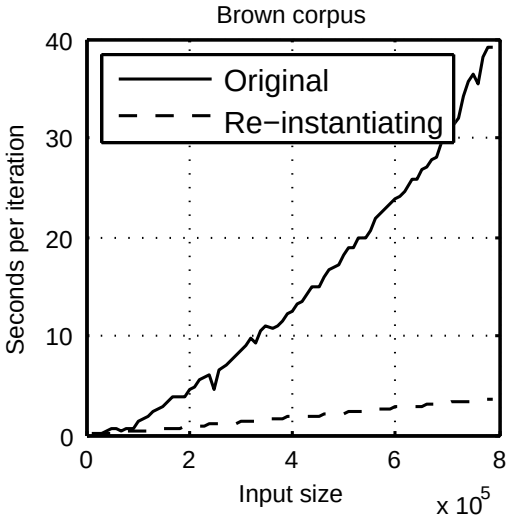


Figure 4.5: Comparison of the ADDREMOVE sampler in the compact representation using either the "original" Gibbs sampler (eqs. (4.1b) & (4.2)) or the Gibbs sampler in the non-compact representation combined with re-instantiating the required seating arrangement using the technique described in Section 4.6) for the Sequence Memoizer model on the Brown corpus. While asymptotically both algorithms have the same complexity, the re-instantiating sampler is significantly faster in practice.

First we re-express A using two sets of variables $z_1,\ldots,z_c$ and $y_1,\ldots,y_c$. Label a table $a \in A$ using the index of the first customer at the table, i.e. the smallest element of a. Let z_i be the number of tables occupied by the first i customers, and y_i the label of the table that customer i sits at. The variables satisfy the following constraints: $z_1 = 1$, $z_c = t$, and either $z_i = z_{i-1}$ in which case $y_i \in [i-1]$ or $z_i = z_{i-1}+1$ in which case $y_i = i$. This gives a one-to-one correspondence between seating arrangements in $\mathscr{A}^t_{[c]}$ and settings of the variables satisfying the above constraints. Consider the following distribution over the variables satisfying the constraints: $z_1,\ldots,z_c$ is distributed according to a Markov network with $z_1 = 1$, $z_c = t$, and edge potentials:

$$f_i(z_i, z_{i-1}) = \begin{cases} i-1-z_i d & \text{if } z_i = z_{i-1}, \\ 1 & \text{if } z_i = z_{i-1}+1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

The joint distribution of $z_{1:c}$ is

$$P(z_{1:c}) = \frac{\prod_{i:z_i=z_{i-1}}(i-1-z_i d)}{Z_d(c,t)}. \tag{4.7}$$

It can be seen via an inductive argument similar to the one in Appendix 5 of [Teh, 2006a] (see also Section A.2 in the appendix) that the normalizing constant is

$$Z_d(c,t) = \sum_{z_{1:c}} \prod_{i:z_i=z_{i-1}} (i-1-z_i d) = S_d(c,t). \tag{4.8}$$

As a base case we have $Z_d(1,1) = S_d(1,1) = 1$, and we further extend the domain of $Z_d(c,t)$ to include $Z_d(0,t) = Z(c,0) = 0$ (for $c > 0, t > 0$), $Z_d(0,0) = 1$, and $Z(c,t) = 0$ (for $t > c$), exactly like the base cases for $S_d(c,t)$ in (A.8).

Now, we can split the summation over $z_{1:c}$ into the cases where (a) $z_c = z_{c-1}$ or (b) $z_c = z_{c-1} + 1$. In case (b) there is no contribution from the last customer/table to the product, so that that the total contribution for that part is $Z_d(c-1, t-1)$. In case (a) the last customer contributes a factor of $c - 1 - td$ to all assignments of $c-1$ customers to t tables, for a total contribution of $(c-1-td)Z_d(c-1,t)$, so that

$$Z_d(c,t) = Z_d(c-1,t-1) + (c-1-td)Z_d(c-1,t) \tag{4.9}$$

which, using the induction hypothesis $Z_d(c-1,t-1) = S_d(c-1,t-1)$ and $Z_d(c-1,t) = S_d(c-1,t)$, exactly matches the recursion for $S_d(c,t)$ given in A.5.

Given $z_{1:c}$, we give each y_i the following distribution conditioned on $y_{1:i-1}$:

$$P(y_i|z_{1:c}, y_{1:i-1}) = \begin{cases} 1 & \text{if } y_i = i \text{ and } z_i = z_{i-1} + 1, \\ \frac{\sum_{j=1}^{i-1} \mathbf{1}(y_j = y_i) - d}{i-1-z_i d} & \text{if } z_i = z_{i-1} \text{ and } y_i \in [i-1]. \end{cases} \tag{4.10}$$

Multiplying all the probabilities together, we see that $P(z_{1:c}, y_{1:c})$ is exactly equal to $\text{CRP}_{ct}(A|d)$ in (4.5): the numerators in the conditional probabilities of the y_i's become the numerator, and the denominators cancel the numerator in $P(z_{1:c})$, while the normalization constant $S_d(c,t)$ is the denominator of $\text{CRP}_{ct}(A|d)$. Thus we can sample A by first sampling $z_{1:c}$ from (4.7), then sequentially sampling each y_i conditioned on the previous ones using (4.10), and finally converting this representation into the equivalent partition A. We use a variant of the forward-filtering-backward-sampling (FFBS) algorithm (see e.g. [Rao and Teh, 2013, Appendix A]) to sample $z_{1:c}$: We reverse the procedure, so that instead of filtering forward (i.e. from $i = 1 \ldots, c$) and sampling backward (from c to 1), we filter backward $i = c, c-1, \ldots 1$, and then sample forward. This avoids numerical underflow problems that can arise when using forward-filtering; filtering backwards avoids these problems by incorporating the constraint that z_c has to equal t into the messages from the beginning. The backward filtering recursion is given by

$$\beta_{z_i}(j) = \sum_{k=1}^{t} \beta_{z_i+1}(k) f_{i+1}(k,j) \quad \text{for } i = c-1, \ldots, 1, j = 1, \ldots, t \tag{4.11}$$

initialized at $\beta_{z_c}(t) = 1$ and the forward sampling sweep then sets $z_i = k$ with probability proportional to

$$f_i(k, z_{i-1})\beta_{z_i}(k) \qquad k \in \{z_{i-1}, z_{i-1}+1\} \tag{4.12}$$

initialized at $z_1 = 1$.[7]

[7] Some computation can be saved by ignoring impossible values of z_i in the computation of β_{z_i}, i.e. using the fact that z_i needs to satisfy $1 \le z_i \le i$ and $t - j \le z_{c-j} \le t$.

CHAPTER 5

Sequence Memoizer: Online Inference

The model construction and inference procedure for the Sequence Memoizer that was presented in the original paper [Wood et al., 2009] is a batch method, i.e. the entire input must be known before the model can be built and inference can start. In [Gasthaus et al., 2010] we presented an incremental model construction and approximate inference procedure that does not require the entire input to be known in advance. The predictive distribution can be approximated at any point in the input, making it suitable for online prediction applications such as data compression (see Chapter 6). Two steps are necessary for this: an incremental algorithm for constructing the context tree, and an incremental inference method.

Inference in the HPYP/SM model amounts to estimating the posterior distribution $P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}|x_{1:i})$, usually by approximating it using samples from this distribution (in the form of settings of the counts $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$ for all observed contexts $\mathbf{u}$). In the batch setting discussed above, these samples were obtained using MCMC sampling after observing the entire input sequence $x_{1:N}$. In the online setting, we require a method that can efficiently infer the sequence of posterior distributions $P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}|x_{1:i})$, as i sequentially increases from 1 to N. In principle, one could simply apply the discussed Gibbs sampling techniques to each element in this sequence separately, but this would be prohibitively expensive for most applications.

Here we propose three approximate inference schemes which sequentially update an approximation to $P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}|x_{1:i-1})$ such that it becomes an approximation to $P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}|x_{1:i})$. The first is simply a sequential version of the "Kneser-Ney approximation" (Section 2.4.2), the second is a sequential Monte

Carlo (SMC) approach, and the third is based on the idea of "fractional customers".

5.1 Online Context Tree Construction

In [Wood et al., 2009] the context tree was constructed by exploiting the fact that the suffix tree of a string corresponds to the prefix tree of its reverse and applying Ukkonen's algorithm [Ukkonen, 1995] to the reverse of the input. As Ukkonen's algorithm processes the string from left to right, the entire string needs to be known ahead of time. Here we argue that using a complex $O(N)$ context/suffix tree construction algorithm such as Ukkonen's algorithm is neither necessary nor beneficial in the setting of sequential prediction in the SM: This is because the computational complexity of the prediction algorithm will be dominated by the cost of computing the predictive probability and updating the posterior distribution over states. Computing the predictive probability (2.45) requires recursively computing (2.45) at each node on the path from the root to the $\mathbf{u}$-node, and there are $|\mathbf{u}|$ such nodes in the worst case (which occurs of $x_{1:i} = s^i$ for some $s \in \Sigma$). As we need to traverse such a path of length $O(N)$ for each symbol in the sequence, the worst case complexity of sequential prediction of a sequence of N symbols using the SM is $O(N^2)$. Further, any sequential inference procedure needs to maintain and update some representation of the distributions $G_{\mathbf{u}}, G_{\pi(\mathbf{u})}, G_{\pi(\pi(\mathbf{u}))}, \ldots$ along path from $\mathbf{u}$ to the root.[1]

The main "trick" used by $O(N)$ suffix tree construction algorithms is that they maintain additional pointers (the so called *suffix links*), which allow the node where the next modification in the tree needs to occur during online construction to be found in constant time. However, for online inference in the SM this is not helpful, as we need access to not only that node but to the entire path from the root to that node.

Therefore, we propose using a "naïve" prefix tree construction algorithm with a worst-case time complexity of $O(N^2)$, which for each insertion traverses exactly the nodes at which computation must be performed to evaluate (2.45), and thus does not increase the asymptotic complexity of the overall algorithm, but only adds a small constant factor.

In practice, the performance is much better than the worst case analysis suggests, as for typical inputs in the language modelling setting the paths in the context tree are very short and the path lengths grow only slowly with the

[1] Note that even in the batch, offline setting the time spent constructing the context tree is typically negligible compared to the time spent performing inference.

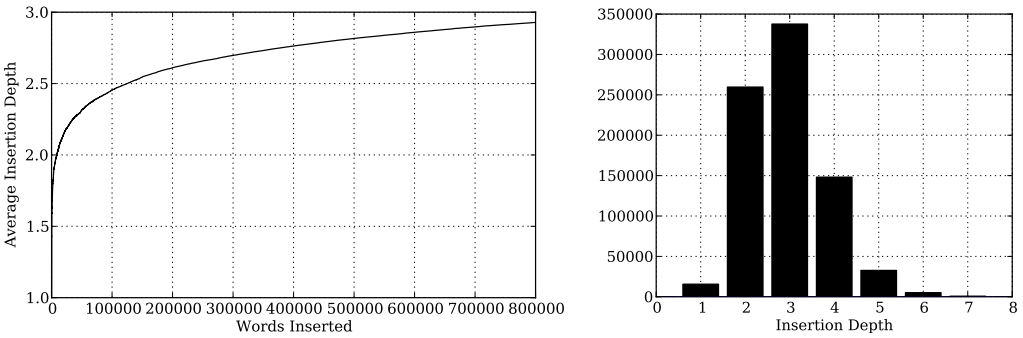


Figure 5.1: Context tree construction node insertion depth statistics on the Brown corpus. The insertion depth of a context is the length of the path from the root to the inserted context, and thus the number of nodes that need to be traversed to make a prediction from that context. Left: Average node insertion depth (i.e. for the i-th context, inserted at depth d_i, we plot $\sum_{j=1}^{i} d_j / i$). Right: Histogram of context insertion depths during tree construction (i.e. histogram of the number of nodes traversed during each context insertion). The histogram is truncated on the right: there are in total 248 insertions at depths ≥ 8, and the maximum depth is 22.

input size (shown in Figure 5.1). Because of this, the use of an $O(N)$ context tree construction algorithm is almost never computationally advantageous, even in the batch setting.

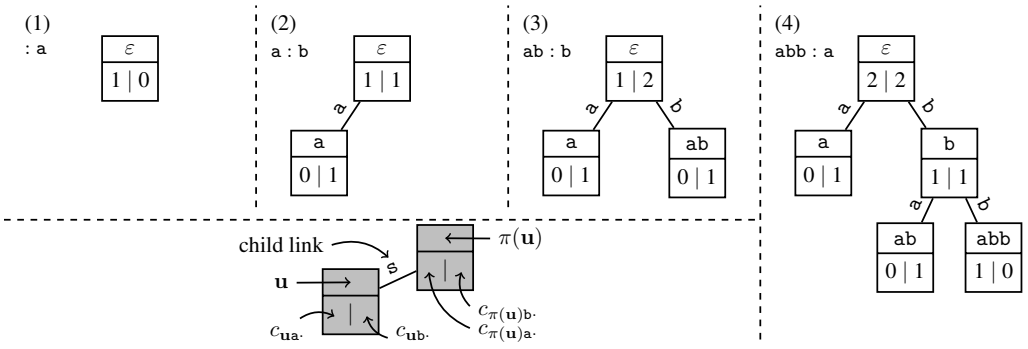


Figure 5.2: Illustration of the tree construction algorithm for the input string abba. Each step shows the state of the tree after inserting the current context and updating the counts for the observed symbol. Panel (1) shows the tree before the insertion of the context a. Because the insertion point for a (the node which has the longest common suffix with a) is the root node, and the empty context ε at the root is a suffix of a, the a-node is inserted directly as a child of the root as shown in panel (2). The same is true for the context ab in (3). When inserting abb to obtain (4) from (3) the algorithm traverses the tree down to the insertion point ab by following the child pointer b. As ab is not a suffix of abb, the ab-node is split into b and ab. The nodes ab and abb are then inserted as children of b with the appropriate child pointers a and b respectively. Shown below the context at each node are the counts $c_{\mathbf{ua}\cdot}$ (left) and $c_{\mathbf{ub}\cdot}$ (right) as produced by the UKN and 1PF update schemes (they coincide for this particular short string).

The naïve algorithm for constructing the context tree proceeds one symbol at a time by inserting the context $x_{1:i}$ into the tree at each step $i = 1, \ldots, N$. To

insert the context $x_{1:i}$ into the tree it proceeds in two stages, which are both depicted in Figure 5.2: First, the tree is traversed by following child pointers from the root to find the *insertion point* for the context $x_{1:i}$ (i.e. the node in the tree that shares the longest suffix with it), say $x_{j:k}$. Second, depending on whether $x_{j:k}$ is a suffix of $x_{1:i}$, it either inserts the $x_{1:i}$ node directly (if it is a suffix) as a child of $x_{j:k}$, or splits the $x_{j:k}$ node into $x_{l:k}$ and $x_{j:k}$ such that $x_{l:k}$ is the longest suffix of both $x_{j:k}$ and $x_{1:i}$ which in turn become children of $x_{l:k}$. Figure 5.2 illustrates the steps taken by this procedure for constructing the context tree for `abb`. Based on this procedure it is easy to see that the context tree for any string has at most $2N$ nodes, as at most 2 new nodes are created during an insertion operation.

5.2 Sequential Kneser-Ney Approximation

Recall from Section 2.4.2 that the Kneser-Ney approximation to inference in the HPYP model sets the counts $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$ such that they correspond to the modified counts in Kneser-Ney smoothing (Section 2.2.3), which amounts to all customers of type s sitting around the same table. In other words, $t_{\mathbf{u}s} = 1$ if $c_{\mathbf{u}s} > 0$ and $t_{\mathbf{u}s} = 0$ otherwise. The counts $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$ for all contexts in the context tree are thus completely determined by the observations (using (2.42)):

$$c_{\mathbf{u}s} = c^{x}_{\mathbf{u}s} + \sum_{\mathbf{v}:\pi(\mathbf{v})=\mathbf{u}} \mathbf{1}[c_{\mathbf{v}s} > 0], \tag{5.1}$$

where $c^{x}_{\mathbf{u}s}$ denote the counts directly resulting from the observations. As $t_{\mathbf{u}s}$ is a deterministic function of $c_{\mathbf{u}s}$, the only data statistics that need to be stored are the counts $c_{\mathbf{u}s}$. Given the counts after processing the prefix $x_{1:i-1}$, one can easily update them to account for the observation x_i in the context $\mathbf{u} = x_{1:i-1}$: First $c_{\mathbf{u}x_i}$ is incremented; if $c_{\mathbf{u}x_i}$ was zero before, the parent count $c_{\pi(\mathbf{u})x_i}$ is incremented; if $c_{\pi(\mathbf{u})x_i}$ was zero before, its parent count $c_{\pi(\pi(\mathbf{u}))x_i}$ is incremented, and so on, until a context with nonzero count for symbol x_i or the root is reached.

When this technique is applied to the Sequence Memoizer model (with modified discounts (3.4)), we refer to it as "Unbounded depth Kneser-Ney" (UKN). The sequential update rule for the counts above is the same as the one used in some variants of the PPM compression algorithm, where this form of update is known as the "shallow update", as it does not require the counts to be updated all the way along the tree (see [Steinruecken, 2014, Section 6.3.3]). The advantage of UKN over the other inference schemes is that it is faster, as no sampling is required, and more memory efficient, as only one count needs

to be stored per context-symbol pair. As we will see in the evaluation, this approximation typically yields competitive predictive performance.

5.3 Sequential Monte Carlo

Another way of performing inference for the posterior distribution over seating arrangements in HPYP/SM models is via sequential Monte Carlo (SMC) methods (variants of which are also known as *particle filtering*). We refrain from giving an introduction to general SMC methods here and refer the reader to [Doucet et al., 2001] and [Cappé et al., 2007] for background. Fearnhead [2004] noted that the sequential nature of the Chinese restaurant process lends itself to sequential inference using such methods and developed an SMC scheme for the Dirichlet process mixture model. Here we show how this idea can be extended to the PYP case with delta function likelihood, as well as to the hierarchical HPYP/SM model.

5.3.1 Sequential Imputation

The underlying SMC algorithm used here and in [Fearnhead, 2004] is the sequential imputation algorithm of Kong et al. [1994]. This is a general algorithm which is applicable to any probabilistic model with observations $x_{1:N}$ and associated latent variables $z_{1:N}$ as long as it is possible to sample from the conditional distribution $\mathrm{P}(z_i|x_i, x_{1:i-1}, z_{1:i-1})$ and the predictive distribution $\mathrm{P}(x_i|x_{1:i-1}, z_{1:i-1})$ can be evaluated. The basic idea is to approximate the posterior distribution over the latent variables $z_{1:i}$ after seeing i observations as a weighted set of M particles $(w_i^{(j)}, z_{1:i}^{(j)})$, $j = 1, \ldots, M$,

$$\mathrm{P}(z_{1:i}|x_{1:i}) \approx \sum_{j=1}^{M} w_i^{(j)} \delta_{z_{1:i}^{(j)}}(z_{1:i}) \tag{5.2}$$

and to update this approximation sequentially by sampling values for z_i (and keeping $z_{1:i-1}$ fixed) as the observations $x_i, i = 1, \ldots, N$ come in. In particular, each particle $z_{1:i-1}^{(j)}$ is extended by sampling

$$z_i^{(j)} \sim \mathrm{P}\left(z_i \,\middle|\, x_i, x_{1:i-1}, z_{1:i-1}^{(j)}\right) \qquad j = 1, \ldots, M \tag{5.3}$$

and the weight updated according to

$$\tilde{w}_i^{(j)} = \tilde{w}_{i-1}^{(j)} \mathrm{P}\left(x_i \,\middle|\, x_{1:i-1}, z_{1:i-1}^{(j)}\right) \tag{5.4a}$$

$$w_i^{(j)} = \left(\sum_k \tilde{w}_i^{(k)}\right)^{-1} \tilde{w}_i^{(j)} \tag{5.4b}$$

with initial weight $\tilde{w}_1^{(j)} = \mathrm{P}(x_1)$. It can be shown that these updates are a form of importance sampling, where the proposal distribution $\mathrm{Q}(\cdot)$ is the sequence of conditional distributions (5.3) and the weights are the standard importance sampling weights $w_i^{(j)} = \mathrm{P}(z_{1:i}^{(j)}|x_{1:i})/\mathrm{Q}(z_{1:i}^{(j)}|x_{1:i})$ [Kong et al., 1994]. Given such a weighted particle approximation, the predictive distribution for a new observation can be approximated as

$$\mathrm{P}(x_{i+1}|x_{1:i}) = \sum_{z_{1:i}} \mathrm{P}(x_{i+1}|x_{1:i}, z_{1:i})\mathrm{P}(z_{1:i}|x_{1:i}) \approx \sum_{j=1}^{M} w_i^{(j)} \mathrm{P}(x_{i+1}|x_{1:i}, z_{1:i}^{(j)}). \quad (5.5)$$

However, it is well known that the weights in the sequential importance sampling scheme above quickly become skewed, ultimately collapsing on a state where all weight is placed on a single particle [Cappé et al., 2007]. In order to alleviate this "weight degeneracy" problem, one typically includes a "rejuvenation" step after computing the weights, in which a new set of particles is chosen from the old ones by duplicating particles with high weight and removing particles with low weight. Various strategies for performing this step have been proposed [Douc et al., 2005], the simplest (and most widely used) of which is known as *multinomial resampling*, where M equally weighted new particles are independently drawn from (5.2).

5.3.2 Particle Filter For A Single PYP

Before turning to the hierarchical PYP models, let us consider a simple single PYP model

$$G \sim \mathrm{PY}(\alpha, d, H) \quad (5.6a)$$

$$x_i \sim G \qquad i = 1, \ldots, N \quad (5.6b)$$

where α and d are fixed and $H(\cdot)$ is some fixed distribution over Σ. We know that this model has an equivalent construction (2.30) in terms of the CRP, where the latent variables are the partition structure $A_{1:K}$. As the prior for this structure—the CRP—is sequential in nature, we can write it in terms of a set of sequential latent variables $z_{1:N}$, one for each observation. One suitable set of variables $z_{1:N}$ to describe $A_{1:K}$ would e.g. be to order the blocks $k = 1, \ldots, K$ by least elements, and let $z_i = k$ denote the block containing i, satisfying the constraint $z_{i+1} \leq 1 + \max z_{1:i=1}$. Using this representation, the conditional distribution of the latent variable z_i used in (5.3) is proportional to prior probability $\mathrm{P}(z_i = k|z_{1:i-1})$ under the CRP, multiplied by $\mathrm{P}(x_{1:i}|z_{1:i})$, which is 0 if the assignment $z_{1:i}$ is inconsistent with $x_{1:i}$, i.e. $\exists i, j : x_i \neq x_j \wedge z_i = z_j$ (customers with different

labels sitting at the same table), and $\propto H(s)^{t_s}$ otherwise. It is thus given by

$$\mathrm{P}(z_i|z_{1:i-1}, x_{1:i-1}, x_i = s) \propto \begin{cases} c_{s\kappa} - d & \text{if } z_i = k(s,\kappa), \kappa = 1, \ldots, t_s \\ (\alpha + td)H(s) & \text{if } z_i = k(s, t_s + 1) \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

where $k(s,\kappa)$ denotes the global table corresponding to the κ-th table in section s, and the counts $c_{s\kappa}, c_s, t_s, t$ (defined as before) are obtained from $x_{1:i-1}$ and $z_{1:i-1}$. Note that this is the same conditional distribution as used in add/remove Gibbs sampler (Algorithm 1). The predictive distribution $\mathrm{P}(x_{i+1}|x_{1:i}, z_{1:i})$ required for computing (5.4) is given by (2.31), which using the notation introduced above becomes

$$\mathrm{P}(x_i = s|z_{1:i-1}.x_{1:i-1}) = \frac{c_s - t_s d + (\alpha + td)H(s)}{\alpha + c}. \tag{5.8}$$

Instead of applying this algorithm to the posterior distribution over seating arrangements, we can also apply it directly to the posterior distribution over table counts t_s (i.e. the COMPACT representation), marginalizing out the seating arrangement, resulting in the joint distribution over $\{c_s, t_s\}$ given by (2.33). The state of the particle filter in this case is just the number of tables of each type $t_{i,s}^{(j)}$ (for the j-th particle after observing the first i symbols). At each step, each particle j is extended by sampling $t_{i,s}^{(j)}$ from

$$P\left(t_{i,s}|t_{i-1,\cdot}^{(j)}, x_{1:i-1}, x_i = s\right) \propto \begin{cases} (\alpha + d\,t_{i-1,\cdot}^{(j)})H(s) & \text{if } t_{i,s} = t_{i-1,s}^{(j)} + 1 \\ c_{i-1,s} - d\,t_{i-1,s}^{(j)} & \text{if } t_{i,s} = t_{i-1,s}^{(j)} \end{cases} \tag{5.9}$$

In other words, the number of tables of type s in particle j is incremented by one with probability proportional to the posterior probability of creating a new table, and is kept the same with probability proportional to the sum of probabilities for joining any table of type s, obtained by summing the first case in (5.7) over $k = 1, \ldots, t_s$. As before, each particle is then weighted by (5.8), which does not depend on the seating arrangement.

As noted by Fearnhead [2004], proposing new particles by *sampling* from (5.7) or (5.9) is rather wasteful, as the state space can be enumerated and there is no advantage to having the same state represented by multiple particles (instead of just one particle with a larger weight). As the number of possible cases in (5.7) and (5.9) is typically small, it usually possible to enumerate all possible extensions for each current particle, resulting in a larger set of putative particles. The weight for each such putative particle is proportional to the weight of its parent, multiplied by the probability of the chosen extension according to

(5.7) or (5.9) and further multiplied by the likelihood of the current observation under the parent particle (5.8). Once this set of putative particles grows beyond some specified maximum size, a resampling step can be employed to select a subset of these particles, e.g. using simple multinomial resampling from the weights or by using the technique proposed in [Fearnhead, 2004], which has the advantage that each putative particle is included at most once in the resulting particle set.[2] Figure 5.3 shows a simulation comparing these algorithms, and it can be seen that enumerating putative particles rather than sampling them leads to significantly smaller variance of the estimates.

As this particle filtering algorithm can yield accurate results even with a small number of particles, it can be useful as a building block within other inference schemes, e.g. as proposal distribution within a Metropolis-Hastings sampler for a hierarchical model. A different approach is taken by the "one-particle particle filter" inference for the Sequence Memoizer that will be discussion in Section 5.3.4, where the sequential imputation technique is used directly on the hierarchical model (and only a single particle is used).

5.3.3 Particle Filter for the HPYP / SM

The sequential Monte Carlo algorithms just described can be extended to the HPYP/SM setting in the same way the add/remove Gibbs sampler (Algorithm 1) for the PYP was extended to the HPYP/SM setting (Algorithm 2): in the hierarchical setting, each particle $z_{1:i}^{(j)}$ consists of all counts ($\{c_{\mathbf{u}sk}, t_{\mathbf{u}s}\}$ (or $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$ in the COMPACT representation) for all nodes $\mathbf{u}$ in the context tree after processing the first i symbols. As before, each iteration starts by extending all particles by sampling from (or enumerating) (5.3), where z_i now denotes all latent variables associated with the i-th observation, throughout the hierarchy. In particular, z_i describes the contribution of x_i to all seating arrangements along the path from $\mathbf{u} = x_{1:i-1}$ to the root in the context tree. In order to sample from this distribution of extended seating arrangements, we can use the ADDCUSTOMER routine from Algorithm 2, which performs the same function in the Gibbs sampler (for the COMPACT version the probabilities (4.1) are used). In summary, each particle $z_{i-1}^{(j)}$ is extended by first seating the customer corresponding to x_i in the $\mathbf{u} = x_{1:i-1}$ restaurant according to (2.24a) (which is (5.7) with $H(s)$ replaced by $G_{\pi(\mathbf{u})}(s)$), and, if this created a new table, recursively repeating the process

[2] When the state space is $\{t_s\}$, particles with exactly the same state can still be created even when the putative particles are generated by enumeration and resampling never duplicates particles: if the previous generation of particles contains both $t_s = k$ and $t_s = k+1$, then the putative particles will contain $t_s = k+1$ twice. These particles can be merged before resampling to improve efficiency further.

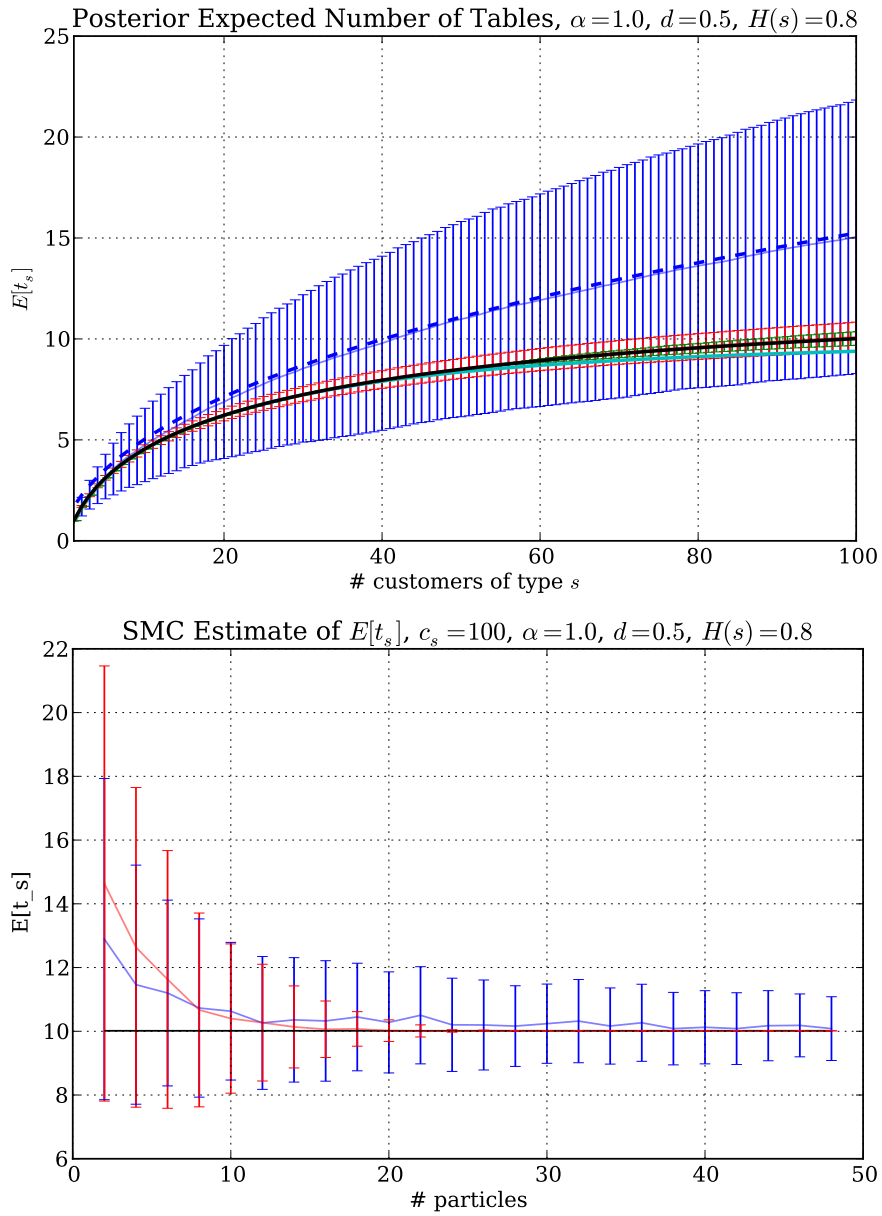


Figure 5.3: Accuracy of the presented SMC algorithms for a single PYP on a synthetic data set: the observations are the same symbol repeated 100 times (i.e. $c_s = 100$ for one symbols s $c_{s'} = 0$ for all other symbols s'). The parameters in this simulation are $H(s) = 0.8, \alpha = 1, d = 0.5$. While the actual results vary with $H(s), \alpha$, and d, the results are qualitatively similar across the parameter range. Error bars show 1 standard deviation and were obtained by repeating the simulation 500 times. Top: Estimate of $E[t_s]$ as a function of c_s. In red is the sampling SMC algorithm on t_s (100 particles), in green the version that enumerates the extensions (20 particles). In cyan is deterministic "beam search" where the 20 particles with the highest weights were kept. In blue is the "one particle" particle filter, the dashed blue line is the "fractional tables" approximation. Bottom: Estimate of $E[t_s]$ at $c_s = 100$ as function of the number of particles. Blue is the sampling version, red the enumerating version. Not shown on here is the particle filter in the full CRP representation: in this particular example, its performance is virtually indistinguishable from the one using the compact representation.

in the parent restaurant corresponding to $G_{\pi(\mathbf{u})}$. The weights (5.4) are given by the hierarchical version of (5.8), given by (2.45). The improvements mentioned in the non-hierarchical setting, i.e. better resampling strategies, enumeration instead of sampling from (5.3), and using the COMPACT representation can also be applied in the hierarchical setting.

5.3.4 One Particle Particle Filter

Based on the observation that the predictive performance of the SM is fairly stable under different settings of the latent variables $\{t_{\mathbf{u}s}\}$ and does not drastically improve by averaging multiple samples (for example in Figure 4.4, the total improvement from the 1PF initialization to the sample average obtained by Gibbs sampling after burn-in is only ≈ 0.02 bits/symbol), an extreme case of the above SMC algorithm using only a single particle $M = 1$ might be an efficient yet effective way of performing inference in the HPYP/SM model. We explored this idea (which we dubbed the "one particle particle filter" (1PF) in Gasthaus et al. [2010]), where we showed that this strategy outperforms the Kneser-Ney approximation and performs not much worse than "proper" inference using Gibbs sampling.

When only a single particle is used, the weighting step (5.4) as well as resampling are unnecessary, so that the resulting algorithm simply consists of repeated calls to ADDCUSTOMER, one for each observation x_i. For concreteness and to show the simplicity of the resulting algorithm, it is given in Algorithm 3.

If we were interested in the posterior distribution over $\{t_{\mathbf{u}s}\}$, this algorithm is obviously not a good idea, as it produces biased samples $\{t_{\mathbf{u}s}\}$, that are not samples drawn from the true posterior distribution, but instead from the sequential proposal distribution (5.3) (see Figure 5.3). However, we are not directly interested in the posterior, but in making predictions for new symbols, and empirically the samples from the 1PF procedure result in predictions that are close to the ones obtained via Gibbs sampling. The results are included in Tables 5.1, 5.2, and 6.1.

Intuitively, this can be understood in the following way: Every time a new observation is added to the model, the counts are incremented for some random distance up the tree. This sharing of counts is the way observations in related context reinforce each other. One of the reasons why single sample/-particle predictive inference works in this model is that much of the predictive ability of the SM comes from this hierarchical sharing of counts. The shared counts result in hierarchical smoothing of the predictive distributions. Averaging over multiple samples simply smooths the predictive distributions

Algorithm 3 HPYP/SM One Particle Particle Filter (1PF)
This algorithm effectively consists of calling ADDCUSTOMER from Algorithm 2 (modified for the COMPACT representation) for each context-symbol pair in the input. Note that if $t^+ = 0$ is sampled in line 6, $t^\star$ will become and stay zero for the remainder of the inner loop in line 8, so that no further changes to $c_{\mathbf{v}s}/t_{\mathbf{v}s}$ will occur and the loop can be exited early. The role of the table-increment variable t^+ as well as the reason for writing the algorithm in this somewhat curious way using a for-loop and the $t^\star$ variable will become apparent in Section 5.4, where a different approximation scheme is devised by replacing the sampling step in line 6 with its expectation.

1: **for** $i = 1, \ldots, N$ **do**
2: Let $\mathbf{u} \leftarrow x_{1:i-1}$ and $s \leftarrow x_i$
3: Insert $\mathbf{u}$ into the context tree
4: Set $c_{\mathbf{u}s} \leftarrow 1, t_{\mathbf{u}s} \leftarrow 1$ ▷ The first customer sits at the first table
5: Initialize $t^\star \leftarrow 1$ ▷ # of tables created in the parent restaurant
6: **for** $\mathbf{v} = \pi(\mathbf{u}), \pi(\pi(\mathbf{u})), \ldots, \varepsilon$ **do** ▷ Contexts along the path to the root
7: Sample $t^+ \in \{0, 1\}$ using (4.1), i.e. with probability proportional to

$$\begin{cases} c_{\mathbf{v}s} - t_{\mathbf{v}s} d_{\mathbf{v}} & t^+ = 0 \\ \alpha_{\mathbf{v}} + d_{\mathbf{v}} t_{\mathbf{v}} . G_{\pi(\mathbf{v})}(s) & t^+ = 1 \end{cases} \tag{5.10}$$

8: $c_{\mathbf{v}s} \leftarrow c_{\mathbf{v}s} + t^\star$ ▷ Increment customers
9: $t^\star \leftarrow t^\star \times t^+$
10: $t_{\mathbf{v}s} \leftarrow t_{\mathbf{v}s} + t^\star$ ▷ Increment tables
11: **end for**
12: **end for**

more.

As an aside, another way of interpreting the HPYP/SM particle filtering algorithm (with $M > 1$) is as an exponentially weighted average online learning strategy optimizing the log loss (see e.g. [Cesa-Bianchi and Lugosi, 2006, Chapter 2]). If no resampling of particles is performed, the described particle filtering algorithm with M particles is equivalent to running M independent copies of the 1PF algorithm and then combining the predictions for the i-th item according to the weights

$$w_i^{(j)} \propto \exp\left(\sum_{k=1}^{i} \log \mathrm{P}(x_k | x_{1:k-1}, z_{1:k-1}^{(j)}) \right) \qquad j = 1, \ldots, M. \tag{5.11}$$

We can thus view the particle filter as an online regret minimization algorithm where the base predictors are obtained using the 1PF algorithm.

5.4 Inference Using Fractional Counts

Given that the different methods for inferring the latent table counts $t_{\mathbf{u}s}$ yield very similar results, and the fact that even a single sample provides good performance, it seems appealing to either use some form of deterministic point

estimate for the $t_{\mathbf{u}s}$ (e.g. $E[t_{\mathbf{u}s}]$ or the MAP assignment) or to derive some other closed-form approximation of the posterior expectation of the predictive distribution.[3]

The Kneser-Ney approximation is one way of obtaining such a point estimate, but it is not derived from the HPYP model, and we have seen that is outperformed by the stochastic 1PF approximation. Another idea for a deterministic point estimate was put forth by Huang and Renals [2007]: Asymptotically, the *prior* expected number of tables grows as a power law in the number of customers (see Section 2.3.4). Based on this observation, they proposed to deterministically set $t_{\mathbf{u}s} = c_{\mathbf{u}s}^{\beta}$, where β in their work is chosen according to the fixed rule $\beta = n_1/(n_1 + 2n_2)$ where n_1 and n_2 are the total number of n-grams with counts exactly one and two respectively (as originally proposed for Kneser-Ney smoothing in [Ney et al., 1994]). They show that this choice performs favorably compared to variants to Kneser-Ney, but does not achieve the performance of the HPYP model where the $t_{\mathbf{u}s}$ are inferred using Gibbs sampling. One simple but unexplored extension to this would be to let each depth have its own parameter $\beta_{|\mathbf{u}|}$, which are optimized on a validation set.

Instead of using the asymptotic growth of t, the prior expectation for the number of tables in a two parameter CRP can be computed in closed form and is given by [Pitman, 2002, Equation 161]:

$$E_{\alpha,d}[t] = \sum_{i=1}^{c} \frac{[\alpha + d]_1^{i-1}}{[\alpha + 1]_1^{i-1}} \tag{5.12}$$

which for $d > 0$ can be written as

$$\frac{[\alpha + d]_1^{n}}{d[\alpha + 1]_1^{n-1}} - \frac{\alpha}{d} \tag{5.13}$$

and in the special case of the Dirichlet process with $d = 0$ reduces to

$$E_{\alpha}[t] = \sum_{i=1}^{c} \frac{\alpha}{\alpha + i - 1} \tag{5.14}$$

which is simply the sum of the probabilities for the (independent) events that the i-th customer creates a new table [Antoniak, 1974]. If each block in the CRP partition is labeled independently with a draw $s \sim H$, then the prior expected value of the number of blocks labeled s is simply $E_{\alpha,d}[t_s] = E_{\alpha,d}[t]H(s)$.

For the Dirichlet process case, one can also compute the *posterior* expected table counts, given by [Antoniak, 1974; Blunsom et al., 2009]:

$$E_{\alpha}[t_s|c_s] = \alpha H(s) \sum_{i=1}^{c_s} \frac{1}{\alpha H(s) + i - 1}. \tag{5.15}$$

[3] Note that in a hierarchical PYP model the predictive distribution (2.45) is not linear in $t_{\mathbf{u}s}$, so that in order to compute the expectation of the predictive distribution under the posterior, it is not sufficient to compute $E[t_{\mathbf{u}s}]$ (even jointly for all $\mathbf{u}$), as the $t_{\mathbf{u}s}$ contribute to the customers $c_{\pi(\mathbf{u})s}$ in the parent restaurant, and these terms are multiplied.

Blunsom et al. [2009] analyzed using this approximation in the context of HDP models for NLP applications. They provided an expression for this exact expectation that can be computed in $O(1)$ time, removing the need for further approximation.[4] They cautioned that this expectation is assuming a fixed base distribution $H(s)$ and showed empirically that when $H(s)$ is not fixed (e.g. in the hierarchical setting), it is not necessarily an accurate approximation. However, Blunsom and Cohn [2011] explored the idea of approximating the posterior expectation of $t_{\mathbf{u}s}$ further (to generate proposals in an MCMC scheme for a PYP-based HMM model), and showed that it performed well in that setting. Here we explore the same idea, based on using "fractional" counts (i.e. $c_{\mathbf{u}s}, t_{\mathbf{u}s} \in \mathbb{R}^+$), in the setting of the HPYP/SM model and show that it is (unreasonably) effective.

Unfortunately, even for the setting with fixed base distribution $H(s)$, no closed-form expression for the posterior number of tables has been derived yet for the general two-parameter case (with $d \neq 0$). This endeavour is complicated by the fact that posterior distribution over the $\{t_s\}_s$ does not factorize over the different types s as it does in the one-parameter case. However, Blunsom and Cohn [2011] have suggested an iterative approximation based on the sequential, one-step probability of creating a new table. Recall that the one particle filter (Algorithm 3) extends its state in each iteration by visiting the restaurants along the path from the current context to the root, sampling the binary "table creation" variables t^+ according to (5.10) and then adding t^+ to the number of tables in the current restaurant and the number of customers in the parent.

The sequential approximation suggested by Blunsom and Cohn [2011] is to replace the sampling of t^+ in line 8 of Algorithm 3 with the computation of its expectation $E[t^+]$ (and setting $t^+ \leftarrow E[t^+]$). In particular, from (5.10) we have

$$E[t^+] = \frac{\alpha_{\mathbf{v}} + d_{\mathbf{v}} t_{\mathbf{v}\cdot} G_{\pi(\mathbf{v})}(s)}{c_{\mathbf{v}s} - t_{\mathbf{v}s} d_{\mathbf{v}} + \alpha_{\mathbf{v}} + d_{\mathbf{v}} t_{\mathbf{v}\cdot} G_{\pi(\mathbf{v})}(s)} \tag{5.16}$$

where the variables on the right and side are updated as in Algorithm 3. While in the particle filter the creation of a new table in a restaurant leads to the insertion of a new customer into the parent restaurant (which in turn can sit at a new table, leading to another customer being inserted into its parent restaurant, and so on), with the approximation (5.16) the "fractional" table contribution t^+ gets send to its parent restaurant as a "fractional customer", where it contributes to another fractional table according to (5.16), scaled by the "size" of the customer being inserted t^+. The variable $t^\star$ in Algorithm 3

[4] Goldwater et al. [2006b] had previously used an approximation to this expectation (due to Antoniak [1974]) in the HDP setting. However, as pointed out by Blunsom et al. [2009], this approximation can be inaccurate in the regime typically encountered in these models. Additionally, the original paper Goldwater et al. [2006b] contained an error (now corrected), dropping an $H(s)$ term from the approximation.

denotes this scaled fractional table contribution coming from the child restaurant. This way the counts along the path are set to their expected values under the one-step posterior distribution, i.e. the resulting counts $t_{\mathbf{u}s}$ are exactly the average counts obtained by running the 1PF algorithm multiple times (see Figure 5.4). We refer to this approximation as the "fractional" tables/customers approximation (FRAC). This idea of using real-valued expectations of binary random variables instead of sampling their values has previously been successfully applied for approximate inference in stochastic neural network models, e.g. in the Helmholtz machine [Dayan et al., 1995].

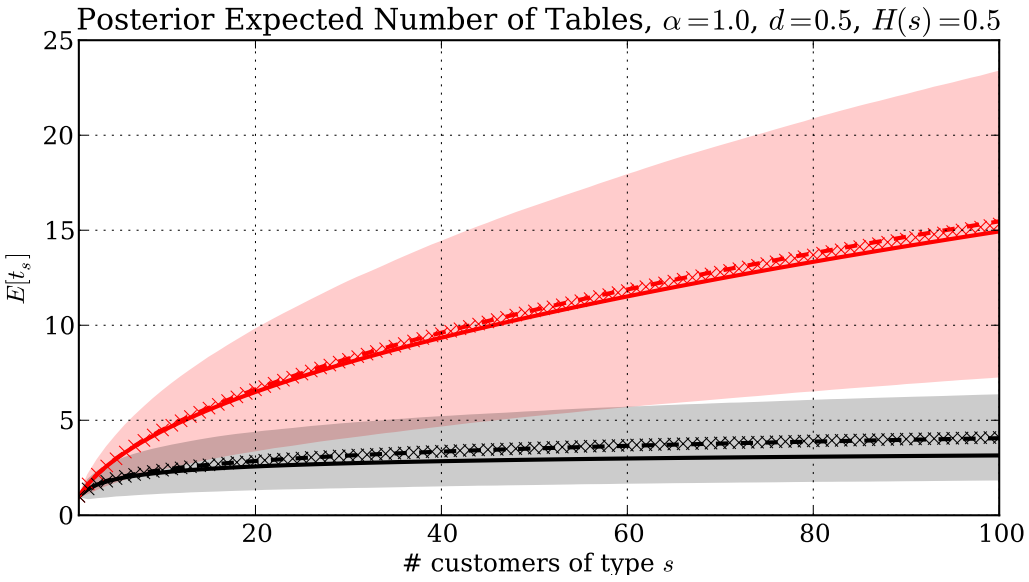


Figure 5.4: Number of tables for multiple runs of the 1PF algorithm and the fractional tables approximation on a two level hierarchy $G_1 \sim \mathrm{PY}(\alpha, d, H)), G_2 \sim \mathrm{PY}(\alpha, d, G_1), x_i \sim G_2$ with $\alpha = 1$ and $d = H(s) = 0.5$ as 100 customers of the same type s are inserted. The solid lines show the true expectation, dashed lines the fractional approximation, and the crosses and shaded area show the mean and 2 standard deviations of the 1PF algorithm (using 5000 runs), respectively. In red are the number of tables in the G_2 restaurant, in black the ones in the G_1 restaurant. Note that the mean under the 1PF algorithm and the fractional tables approximation coincide.

In terms of the simpler single PYP model (5.6), the approximation we are making is to turn the exact one-step-ahead formula

$$E\left[t_s^{[1:i]} \middle| \left\{c_s^{[1:i-1]}, t_s^{[1:i-1]}\right\}_s\right] = t_s^{[1:i-1]} + E\left[t^+ \middle| \left\{c_s^{[1:i-1]}, t_s^{[1:i-1]}\right\}_s\right] \mathbf{1}[x_i = s], \tag{5.17}$$

where the superscript $[1:i]$ means conditioning on the first i observations $x_{1:i}$, and the binary random variable t^+ is defined as

$$\mathrm{P}(t^+ = k | \{c_s, t_s\}_s) = \begin{cases} \frac{c_s - t_s d}{(c_s - t_s d) + (\alpha + dt.H(s))} & k = 0 \\ \frac{\alpha + dt.H(s)}{(c_s - t_s d) + (\alpha + dt.H(s))} & k = 1, \end{cases} \tag{5.18}$$

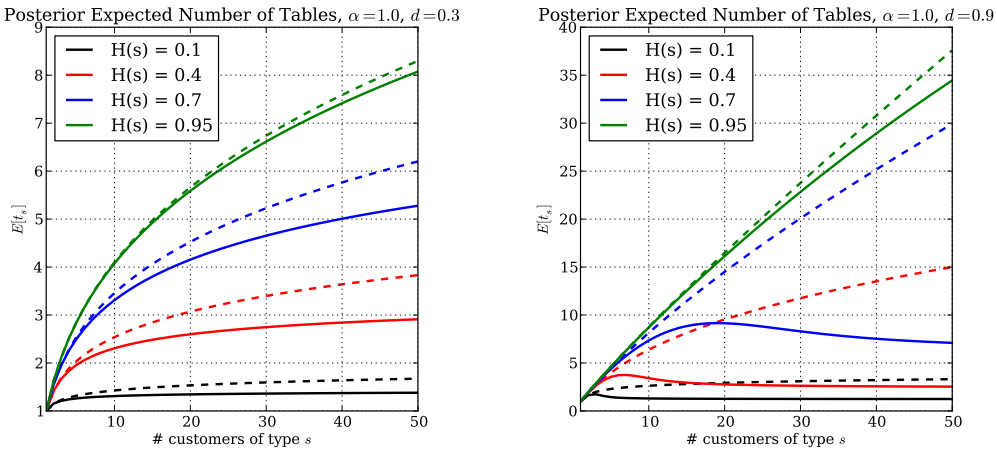


Figure 5.5: One-step expectation "fractional tables" approximation to the posterior expected number of tables in a two parameter CRP. The approximation is shown as dashed lines and the true expectations are shown as solid lines. On the x-axis are the number of customers c_s of type s with probability $H(s)$ under the base distribution. The discount parameters are chosen as $d = 0.3$ for the left panel and $d = 0.9$ for the right panel. The different colours show the effect of different base probabilities $H(s)$. For these experiments, all observations are of the same type, such that $c_\cdot = c_s$ and $t_\cdot = t_s$. The quality of the approximation deteriorates for intermediate values of $H(s)$ as d gets closer to 1.

into an approximation by iteratively replacing $t_s^{[1:i]}$ in the RHS of (5.17) by its expectation, starting with $t_s^{[1:0]} = 0$ and setting $t^+ = 1$ for the first i such that $x_i = s$ (for all s, i.e. the first customer of a given type s always sits at the first table).

The accuracy of this approximation depends on the mismatch between the base distribution $H(s)$ and the empirical distribution of the observations, the order in which the observations are incorporated into the approximation, and the discount parameter. Figure 5.5 compares the accuracy of this approximation for different discount parameters and different base distributions. In Figure 5.6 we demonstrate the effect of this approximation in a simple three-level PYP hierarchy. When d is large, the approximation can significantly overestimate the true expectation.

The approximation can either be used as a starting point for a proposal distribution in a batch MCMC inference scheme (as originally proposed in [Blunsom and Cohn, 2011]), or it can be used directly in the online setting. We demonstrate the effectiveness of this approximation when applied to online prediction in the context of data compression (Chapter 6) where it is shown to outperform the other approximate inference schemes with comparable computational complexity (cf. Table 6.1).

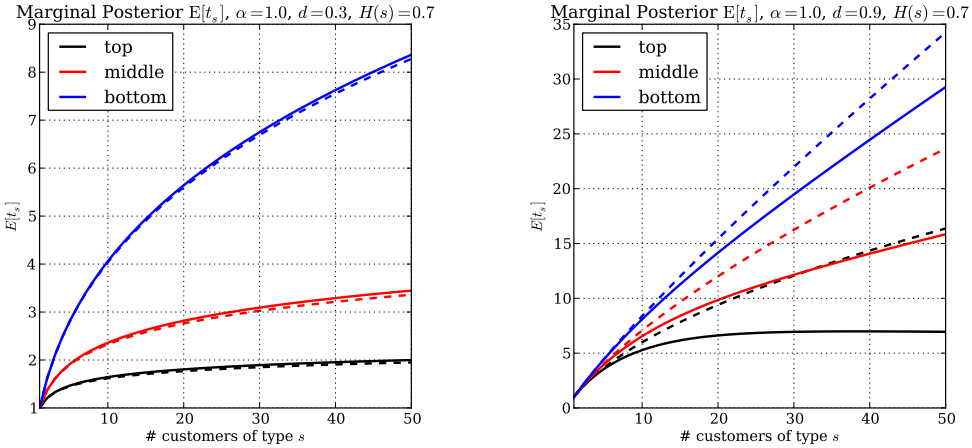


Figure 5.6: Hierarchical "fractional tables" approximation to the posterior marginal expected number of tables in hierarchy of three PYPs. The hierarchy is $G_1 \sim \mathrm{PY}(\alpha, d, H)$, $G_2 \sim \mathrm{PY}(\alpha, d, G_1)$, $G_3 \sim \mathrm{PY}(\alpha, d, G_2)$, $x_i \sim G_3$ with $\alpha = 1$ and $d = 0.3$ (left) or $d = 0.9$ (right). Dashed lines show the approximation, true values are shown as solid lines. On the x-axis are the number of customers c_s in the bottom restaurant. The different colours show the different levels or the hierarchy. For small to intermediate values of d the quality of the approximation is very good, but as in the non-hierarchical case it deteriorates for intermediate values of $H(s)$ as d gets closer to 1.

5.5 Local Optimization

Inspired by the good empirical performance of the approximation to the marginal expectations of $t_{\mathbf{u}s}$, we considered sequentially and locally optimizing $t_{\mathbf{u}s}$ wrt. its conditional posterior distribution (2.33), keeping all other values, including the probability under the base distribution, fixed. After a new customer of type s is inserted into context $\mathbf{u}$, the number of tables $t_{\mathbf{u}s}$ is optimized, followed by an optimization of $t_{\pi(\mathbf{u})s}$ and so on. However, this approach turned out to be not very successful. On the one hand, this local optimization provides a poor approximation to the global MAP assignment in the hierarchical setting. On the other hand, while the predictive performance is on par with the KN approximation, it is computationally much more expensive as it scales quadratically with $c_{\mathbf{u}s}$ due to the computation of the Stirling numbers. An improvement to this local optimization is to optimize the joint posterior distribution (2.43) of t_s along a path, e.g. $(t_{\mathbf{u}s}, t_{\pi(\mathbf{u})s}, t_{\pi(\pi(\mathbf{u}))s}, \ldots, t_{\pi^K(\mathbf{u})s})$, where the accuracy (and complexity) of the approximation can be increased by increasing the path length K. However, experiments with $K = 1$ (pairs of nodes) did not lead to promising results and showed that this approach is computationally too expensive in practice. Figure 5.7 illustrates the behavior of these approximations for a simple three-level hierarchy.

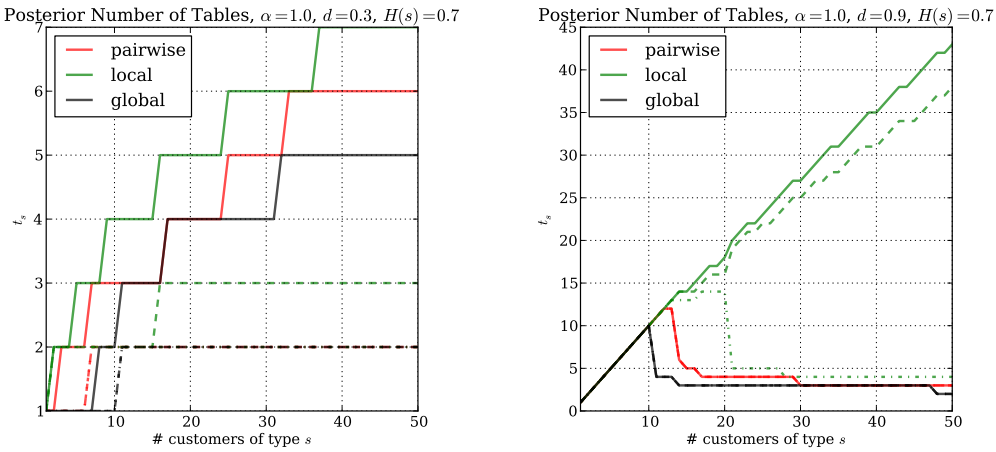


Figure 5.7: Local optimization approximation to the posterior conditional number of tables in hierarchy of three PYPs. The hierarchy is $G_1 \sim \mathrm{PY}(\alpha, d, H)$, $G_2 \sim \mathrm{PY}(\alpha, d, G_1)$, $G_3 \sim \mathrm{PY}(\alpha, d, G_2)$, $x_i \sim G_3$ with $\alpha = 1$ and $d = 0.3$ (left) or $d = 0.9$ (right). On the x-axis are the number of customers c_s in the bottom restaurant. The different line types show the different levels or the hierarchy (solid: bottom; dashed: middle; dash-dotted: top), while the different colors show the different approximations (global: global optimum; pairwise: bottom up optimization using pairs of nodes; local: single node conditional posterior optimization).

5.6 Re-Instantiation

One complication that we have ignored thus far arises when combining these online inference schemes with the online context tree construction for the Sequence Memoizer model: during tree construction, context splits can occur, creating a new node for a context $\mathbf{v}$ whose seating arrangement $A_{\mathbf{v}}$ or counts $\{t_{\mathbf{v}s}\}_s$ associated with the CRP representation of $G_{\mathbf{v}}$ need to be determined. This re-instantiation of the CRP representation of a distribution that had previously been marginalized out (i.e. where the context is not explicitly represented in the context tree) is also necessary when making predictions for contexts whose longest shared suffix with the training sequence is not explicitly represented in the context tree, and the technique for handling that case discussed in Section 3.4 can also be applied here. When a split occurs as position i, there are four nodes involved: the new, longer context being inserted $\mathbf{u} = x_{1:i-1}$, the node being split $\mathbf{u}'$, the new, shorter context resulting from the split $\mathbf{v}$ which becomes the new parent of $\mathbf{u}$ and $\mathbf{u}'$, as well as the parent $\pi(\mathbf{v})$ of $\mathbf{v}$ (which was the parent of $\mathbf{u}'$ before the split). In the example in Figure 5.2 these correspond to the contexts `abb`, `ab`, `b`, and ε, respectively.

As $x_i = s$ is the first symbol observed in context $\mathbf{u}$, it will correspond to the first customer sitting alone at the first table, i.e. we initialize $c_{\mathbf{u}s} = 1$, $t_{\mathbf{u}s} = 1$ for $s = x_i$ and $c_{\mathbf{u}s} = 0$, $t_{\mathbf{u}s} = 0$ otherwise.

Let us consider the state after splitting while inserting the context $\mathbf{u} = x_{1:i-1}$, but before the symbol $x_i = s$ is inserted (i.e. just before line 4 in Algorithm 3). In the simplest case, the Kneser-Ney approximation, all customers of a type s sit at the same table, so that after the split, using (2.42) we have

$$c_{\mathbf{v}s} = t_{\mathbf{u}'s} = \mathbf{1}[c_{\mathbf{u}'s} > 0] \qquad t_{\mathbf{v}s} = \mathbf{1}[c_{\mathbf{v}s} > 0] = \mathbf{1}[t_{\mathbf{u}'s} > 0] \tag{5.19}$$

and all other counts remain unchanged.

When using the particle filter (or the special case with a single particle), we can re-instantiate the counts / seating arrangement for the $\mathbf{v}$ restaurant as described in Section 3.4, by sampling a fragmentation of the tables in the $G_{\mathbf{u}'}$ restaurant from the CRP. If the COMPACT representation is used, this requires sampling the seating arrangement using (4.5) first.

When using "fractional counts" (Section 5.4), it is not clear how to perform the required approximate fragmentation operation on the non-integer counts resulting from this approximation. To get around this we make another approximation and assume that all fractional customers coming from the $G_{\mathbf{u}'}$ restaurant sit at their own fractional tables in $G_{\mathbf{v}}$, i.e. we set

$$c_{\mathbf{v}s} = t_{\mathbf{u}'s} \qquad t_{\mathbf{v}s} = t_{\mathbf{u}'s} \tag{5.20}$$

and leave all other counts unchanged. This setting satisfies the constraints (2.42) along the path $G_{\pi(\mathbf{v})} \to G_{\mathbf{v}} \to G_{\mathbf{u}'}$ and leads to good performance empirically.

When making predictions in a new, unseen context $\mathbf{u}$ that is not explicitly represented in the context tree, one can either proceed as above and first re-instantiate the restaurant for its parent context $\mathbf{v}$ and then make predictions using $G_{\mathbf{v}}$, which would be the correct thing to do. However, we found that in practice making the additional computational effort of re-instantiating $G_{\mathbf{v}}$ might not be necessary, as making predictions using the parent context $\pi(\mathbf{u})$ yields similar predictive performance and does not require the (relatively expensive) operation of sampling and fragmenting a seating arrangement. Predictive performance using both strategies is shown in Table 5.1.

5.7 Hyperparameters $\alpha_{\mathbf{u}}$ and $d_{\mathbf{u}}$

In this section we will turn to the hyperparameters $\alpha_{\mathbf{u}}$ and $d_{\mathbf{u}}$, by first discussing an extended hyperparameter range that we proposed in [Gasthaus and Teh, 2010], and then outlining several strategies for setting them.

5.7.1 Extended Hyperparameter Range

In the original paper on the Sequence Memoizer [Wood et al., 2009], we proposed setting all the concentration parameters $\alpha_{\mathbf{u}}$ to zero. Though limiting the flexibility of the model somewhat, this allowed us to take advantage of a simplified version of the coagulation and fragmentation properties of PYPs (Section 2.3.5) to marginalize out all but a linear number (in N) of restaurants from the hierarchy, as described in Section 3.3.

However, $\alpha_{\mathbf{u}} = 0$ is not the only possible setting that allows the coagulation/fragmentation result to be applied. In [Gasthaus and Teh, 2010] we proposed the following enlarged family of hyperparameter settings: In the hierarchical prior underlying the Sequence Memoizer (3.2), let $\alpha_\varepsilon = \alpha > 0$ be free to vary at the root of the hierarchy, and set each $\alpha_{\mathbf{u}} = \alpha_{\sigma(\mathbf{u})} d_{\mathbf{u}}$ for each $\mathbf{u} \in \Sigma^* \setminus \{\varepsilon\}$. The discounts $d_{\mathbf{u}}$ can vary freely.

In addition to more flexible modelling, this also partially mitigates the overconfidence problem that occurs when the same symbol is repeated many times in the input (as observed in the data compression setting in [Gasthaus et al., 2010]). To see why, notice from (2.45) that the predictive probability is a weighted average of predictive probabilities given contexts of various lengths. Since $\alpha_{\mathbf{v}} > 0$, the model gives higher weights to the predictive probabilities of shorter contexts (compared to $\alpha_{\mathbf{v}} = 0$). These typically give less extreme values since they include influences not just from the sequence of identical symbols, but also from other observations of other symbols in other contexts.

Hyperparameter settings of this form also retain the coagulation and fragmentation properties, which allow us to carry out the marginalization that makes efficient inference in the Sequence Memoizer possible. To see this, consider the three level chain

$$G_0 \sim \mathrm{PY}(\alpha, d_0, H) \quad G_1 \mid G_0 \sim \mathrm{PY}(\alpha d_1, d_1, G_0) \quad G_2 \mid G_1 \sim \mathrm{PY}(\alpha d_1 d_2, d_2, G_1), \tag{5.21}$$

which uses hyperparameters of the form described. Then using Theorem 1 from Section 2.3.5, we can first marginalize out G_1 to obtain $G_2 \mid G_0 \sim \mathrm{PY}(\alpha d_1 d_2, d_1 d_2, G_0)$ and then apply it again to marginalize out G_0 to obtain $G_2 \mid H \sim \mathrm{PY}(\alpha d_1 d_2, d_0 d_1 d_2, H)$. Of course we could alternatively first marginalize out G_0 to obtain $G_1 \sim \mathrm{PY}(\alpha d_1, d_0 d_1, H)$ and then marginalize out G_1 to obtain the same marginal distribution for G_2. So by marginalizing out nodes recursively we can obtain the same context tree representation as in the original SM paper. Note that the concentration parameter for a given distribution does not change through this marginalization, and that the discount parameters change in exactly the same way as in the $\alpha_{\mathbf{u}} = 0$ case described in Section 3.4,

	Particle Filter only		**Gibbs (1 sample)**		**Gibbs (50 sample avg.)**		**Online**	
α	Fragment	Parent	Fragment	Parent	Fragment	Parent	PF	Gibbs
0	8.45	8.41	8.44	8.41	8.43	8.39	8.04	8.04
1	8.41	8.39	8.40	8.39	8.39	8.38	8.01	8.01
3	8.37	8.37	8.37	8.37	8.35	8.35	7.98	7.98
10	8.33	8.34	8.33	8.33	8.32	8.32	7.95	7.94
20	8.32	8.33	8.32	8.32	8.31	8.31	7.94	7.94
50	8.32	8.33	8.31	8.32	8.31	8.31	7.95	7.95

Table 5.1: Average log-loss on the Brown corpus (test set) for different values of α, different inference strategies, and different modes of prediction. Inference is performed by either just using the particle filter or using the particle filter followed by 50 burn-in iterations of Gibbs sampling. Subsequently either 1 or 50 samples are collected for prediction. Prediction is performed either using fragmentation or by predicting from the parent node. The final two columns labelled *Online* show the results obtained by using the particle filter on the test set as well, after training with either just the particle filter or particle filter followed by 50 Gibbs iterations. Non-zero values of α can be seen to provide a significant increase in performance, while the gains due to averaging samples or proper fragmentation during prediction are small.

i.e. the resulting discounts are the product of all discount parameters along the marginalized path.

We performed a set of experiments using the re-instantiating sampler to evaluate the effect of the non-zero concentration parameter. The results are shown in Table 5.1. Predictions with the SM can be made in several different ways. After obtaining one or more samples from the posterior distribution over customers and tables (either using particle filtering or Gibbs sampling on the training set) one has a choice of either using particle filtering on the test set as well (online setting), or making predictions while keeping the model fixed. One also has a choice when making predictions involving contexts that were marginalized out from the model: one can either re-instantiate these contexts by fragmentation or simply predict from the parent (or even the child) of the required node (a further approximation to increase computational efficiency). While one ultimately wants to average predictions over the posterior distribution, one may consider using just a single sample for computational reasons.

5.7.2 Hyperparameter Optimization

Experience with HPYP models and the Sequence Memoizer has shown that getting the hyperparameters in the correct range is crucial for good performance, and fine-tuning them to the data at hand leads to additional improvements. In fact, Steinruecken [2014] suggests that the particular choice of discount parameters $d_{|\mathbf{u}|}$ that increase towards one with greater context length $|\mathbf{u}|$ is responsible

for the SM model's performance not deteriorating when long contexts are used, an effect that has been observed e.g. when extending PPM to unbounded context lengths in PPM* [Cleary and Teahan, 1997]. In the following we will outline some strategies for optimizing or inferring the SM hyperparameters in the offline as well as in the online setting.

Sampling Hyperparameters

In the original work on the HPYP language model, Teh [2006b] suggested sampling the hyperparameters using an auxiliary variable MCMC scheme, interleaved with sampling the seating arrangements. By placing Beta priors on the discount parameters $d_{\mathbf{u}}$ and Gamma priors on the $\alpha_{\mathbf{u}}$, conjugate auxiliary variables can be introduced that when sampled lead to simple conditional distributions for $d_{|\mathbf{u}|}$ and $\alpha_{|\mathbf{u}|}$ that can easily be sampled from (see [Teh, 2006b, Appendix C] for details).

For the Sequence Memoizer, where the discounts $d_{\mathbf{u}}$ are products of the underlying discount parameters $d_{|\mathbf{u}|}$, this auxiliary variable scheme is no longer viable, which is why in [Wood et al., 2009] Metropolis-Hastings updates with Gaussian proposals centered at the current value were used, with a uniform prior placed on the discount parameters $d_{|\mathbf{u}|}$.

The main advantage of sampling the hyperparameters (as opposed to optimizing them) is that no separate validation set is required, so that the full train and validation sets can be used for both estimating the counts as well as inferring the posterior distribution over the hyperparameters. However, sampling is difficult in the online setting and can be slow in the offline setting, and optimizing the hyperparameters, either on a validation set or online, is a viable alternative.

Optimizing Parameters on a Validation Set

For traditional smoothing methods such as interpolated Kneser-Ney smoothing, it is standard practice to optimize the hyperparameters by minimizing cross-entropy on a validation set using standard numerical optimization techniques [Chen and Goodman, 1999]. The same technique was also applied to the HPYP setting by Teh [2006a], who found that optimizing the hyperparameters for the counts obtained by the Kneser-Ney approximation and then using those parameters for the HPYP model outperformed sampling the hyperparameters (for a reduction in perplexity from 103.8 to 101.9 on the AP news corpus), despite effectively optimizing the parameters of the "wrong" model. Note that it is necessary to set aside part of the available data as validation set

for this approach – batch optimization of the hyperparameters on the training set leads to undesirable results. This means that in practice there is a trade-off between using more data to obtain the counts and using data to optimize the hyperparameters.

For the Sequence Memoizer, the same approach can be applied. We experimented with several variants, differing in how inference of the table counts and the parameter optimization are interleaved and whether the Kneser-Ney approximation is used to determine the counts or not. When the KN approximation is used, the setting of the hyperparameters does not influence the latent variables in the model – these are set deterministically independently of the discount and concentration parameters. Because of this, it is not necessary to re-infer the latent variables when the parameters are changed, so that they can be computed once and then remain fixed. The cross-entropy objective function is then just a function of the discount and concentration parameters and can be optimized e.g. using (quasi) Newton methods such as L-BFGS [Nocedal and Wright, 2006].

Similarly, instead of obtaining the latent counts using the Kneser-Ney approximation, they can be obtained using the 1PF algorithm or the fractional tables approximation, fixed, and the resulting function of the hyperparameters optimized. More interestingly, one can interleave the optimization of the hyperparameters with Gibbs sampling. In such a scheme, the latent counts are first initialized using the KN or 1PF approximations, and each subsequent iteration first optimizes the loss on the validation set for some number of steps, holding the counts fixed, followed by some number of rounds of Gibbs sampling, holding the hyperparameters fixed. Following a number of such iterations one can either collect samples directly, or fold the validation set into the training set, fix the optimal hyperparameters, and run the Gibbs sampler again. Teh [2006a] also mentioned such a scheme, but deemed it too costly. In our experiments, one iteration consists of five steps of L-BFGS updates to the hyperparameters, followed by a single sweep of Gibbs sampling, takes $\approx$ one minute per iteration and converges after less than 100 iterations.

Table 5.2 compares two fixed hyperparameter settings (uniform and "standard") against optimizing hyperparameters for fixed counts (obtained using the KN approximation) and interleaving hyperparameter optimization on the validation set with Gibbs sampling of the counts obtained from the training on the Brown corpus. The uniform hyperparameter setting is $\alpha = 0$ and $d_{|\mathbf{u}|} = 0.5$, while the "standard" hyperparameter setting is the one used in [Gasthaus et al., 2010; Gasthaus and Teh, 2010] and amounts to $\alpha = 0$,

Inference	Uniform	Standard	Opt. Uniform	Opt. Standard
KN	8.92/8.86	8.57/8.52	8.40/8.34	8.40/8.34
1PF	8.85/8.79	8.52/8.46	8.34/8.29	8.37/8.31
1PF + Gibbs	8.33/8.77	8.52/8.46	8.32/8.25	8.32/8.25
1PF + Gibbs (avg)	8.77	8.45	8.24	8.24

Table 5.2: Effect of hyperparameter optimization on prediction performance on the Brown corpus. Shown is the average log-loss on the validation set / test set (test set only for the last row). The first two columns correspond to the initial, non-optimized hyperparameter settings (see text). The last two columns correspond to the optimized hyperparameters, after initializing the optimization with either setting. The first two rows correspond to fixing the latent counts using either the KN or the 1PF approximation, the last two rows correspond to the alternating optimization/Gibbs sampling scheme. The last row shows the loss obtained by averaging the predictions for all samples obtained using this procedure.

$d_{[0:9]} = (0.05, 0.7, 0.8, 0.82, 0.84, 0.88, 0.91, 0.92, 0.93, 0.94)$, $d_\infty = 0.95$. We refrain from folding the validation set into the training set after optimization here in order to not conflate the effects and to make the results comparable to previous work. Three effects can be observed: Firstly, optimizing the hyperparameters yields significant performance improvements (the best loss obtained for fixed hyperparameters in Table 5.1 is 8.31 bits/word). Secondly, the interleaved procedure yields similar results for both initializations. Thirdly and somewhat surprisingly, when the 1PF approximation is used, the latent counts obtained using the uniform hyperparameter initialization lead to better final performance after optimization than the "standard" parameters. In all cases, the resulting parameter values are similar. In particular, the ones leading to the best result are $\alpha = 6.33$, $d_{[0:9]} = (0.998, 0.76, 0.92, 0.96, 0.99, 0.99, 0.999, 0.999, 0.999, 0.91)$ and $d_\infty = 0.997]$ (the value 0.999 was used as an upper bound during the optimization). Fixing these parameters and folding the validation set into the training set one obtains a loss of 7.94 bits/word on the test set (averaging 100 samples).

To test the transferability of these optimal values we used these settings and computed the log-loss for the AP news corpus (again folding in the validation set and averaging 100 samples). Using the optimal values obtained on the Brown corpus yields a log-loss of 6.66 bits/symbol, whereas optimizing the hyperparameters using the same interleaved procedure directly on the AP corpus yields a slightly better 6.62 bits/symbol.

Online Parameter Learning using SGD

In the online setting, in addition to building the context tree incrementally and online inference for the counts $c_{\mathbf{u}s}$ and $t_{\mathbf{u}s}$ we may also want to simultaneously

optimize the concentration and discount parameters of the model in an online fashion. One way of doing this is to interleave updates for the counts (KN, 1PF, or the fractional approximation) with a stochastic gradient descent update to the hyperparameters: After receiving a new input (but before updating the counts), we compute the gradient of the logarithm of the predictive probability (2.45) with respect to the parameters. The required gradients can be computed recursively in a single pass along the path from the root to the node where the prediction is made and thus the computational overhead for this computation is small (it scales as the product of the number of parameters and the path length). We then update the counts using either of the described methods, and finally update the parameters by taking a small step in the direction of the gradient, thus increasing the probability of observing the same symbol in the same context. If the counts were fixed, this update corresponds exactly to standard stochastic gradient descent in the online learning setting. For example, the PAQ8 family of compression algorithms update their parameters in this way after observing a new symbol [Knoll, 2011]. However, as the counts are not fixed in our case, the resulting algorithm is not straightforward to analyze, and this should be seen as a heuristic requiring further study. One way of interpreting this procedure is as an online EM algorithm [Cappé and Moulines, 2009; Del Moral et al., 2010], where the expectations with respect to the posterior distribution of the latent variables are approximated rather crudely using a single (approximate) sample.

We found empirically that optimizing the parameters in this way using a small (fixed) learning rate did only marginally improve performance when initialized from hand-tuned parameter values, but did achieve comparable results when all discount parameters where uniformly initialized to 0.5 (see Table 6.1 for results predictive performance in the compression setting using this technique).

CHAPTER 6

The Sequence Memoizer for Data Compression

One of the fundamental results in information theory is that the minimal number of bits required to code a sequence of symbols $x_{1:T} \sim \mathrm{P}(x_{1:T})$ is $-\log_2 \mathrm{P}(x_{1:T})$, and the Kullback-Leibler divergence KL[P‖ Q] measures the number of extra bits required on average if Q is used for coding instead of the true distribution P. It is hence not surprising that many statistical techniques for lossless data compression are based on finding good approximations to the underlying data-generating distribution.

Clever adaptive entropy coding techniques such as arithmetic coding [Witten et al., 1987] have made it possible to directly turn any sequential probabilistic prediction algorithm that at any time point t outputs a predictive distribution over the next symbol $\mathrm{P}_t(x_t)$ into a practical lossless compression algorithm that is asymptotically optimal (see e.g. [MacKay, 2003, Chap. 6]). Any improvement in a predictive model thus directly translates into better compression performance. The symbols in this case (i.e. the underlying alphabet Σ) are typically either the bytes ($\Sigma = \{0, 1, \ldots, 255\}$ or bits ($\Sigma = \{0, 1\}$).

While some of the currently commonly used lossless compression programs such as `gzip` or `bzip2` are not (directly) based on this idea, other mainstream compression programs are, for example those in the RAR family, which is based on a variant of PPM (see below). However, many of the best currently known general-purpose compressors (in terms of compression ratio) are based on this principle.[1]

[1] See for example the leaderboard of compression algorithms on a text compression benchmark maintained by Matt Mahoney at `http://mattmahoney.net/dc/text.html`. Almost all of the top contenders are based on adaptive entropy coding, using either adaptive ensembles of predictors (know as "context mixing" in the compression community) or variants of PPM as the underlying predictive model.

Given that the Sequence Memoizer performs well as a sequence model in the language model setting, it is natural to assess its performance as a predictive model for lossless data compression. The main ingredients that make this possible, namely online context tree construction and online inference, were developed in the last sections so that we focus on a more pragmatic description of the resulting algorithm here. This work was first published in [Gasthaus et al., 2010].

6.1 Related Work

A large amount of work has been done on predictive models for data compression (see e.g. [Mahoney, 2013] and references therein). Conceptually very similar to the SM is the PPM family of algorithms [Cleary and Witten, 1984; Moffat, 1990; Bunton, 1997] (see [Cowans, 2006; Steinruecken, 2014] for a review from a probabilistic modelling perspective), which are based on incrementally estimating a byte-level n-gram model based on a back-off scheme. There are several variants of PPM that differ in the way the symbol and back-off probabilities are estimated. The PPM-B and PPM-D variants make use of absolute discounting and are thus similar to non-interpolated Kneser-Ney smoothing (see Section 2.2.3) and the prediction rules resulting from the HPYP/SM model [Cowans, 2006, Sec. 2.3.7]. The PPM* [Cleary and Teahan, 1997] variant uses a similar (though not identical) compressed trie data structure to efficiently handle unbounded context lengths. However, based on the experiments presented in that paper, it appears that PPM is unable to make use of the information contained in longer contexts, though some improvements have been made in the PPM-Z variant by modifying the probability estimation rule [Bloom, 1998]. Steinruecken [2014] explored the curious discrepancy between the predictive performance of PPM* and the SM, and found that depth-dependent discount parameters used in the SM are likely the reason for its superior performance.

The context tree weighting (CTW) algorithm [Willems et al., 1995] was originally introduced for data compression, where it is used to make bit-level predictions. While the original CTW algorithm required an upper bound on the maximum context length, it has also been extended to remove that bound [Willems, 1998], again using similar though not identical compressed trie data structure. Context Tree Switching (CTS) [Veness et al., 2012] and its recent extension Skip-CTS [Bellemare et al., 2014], perform CTW's "double mixture" (over trees and node parameters) over a larger class of models and have been shown to outperform CTW on data compression benchmarks, and have also

successfully been applied to other prediction tasks.

The PAQ family of compression algorithms are based on the idea of *context mixing*, which makes predictions using an adaptively weighted ensemble of base predictors [Mahoney, 2005], where the mixing weights are chosen based on context and optimized sequentially. PAQ8, a modern variant, employs a neural network trained using online stochastic gradient descent on the prediction error to perform the weighting. Numerous variants exist, which differ in the base predictors used, the exact weighting architecture and the various pre- and postprocessing techniques employed. Most variants are only documented in the software itself, though attempts have been made to document, study, and improve these methods [Knoll, 2011; Knoll and de Freitas, 2012]. Given the recent success of ensemble methods for many prediction tasks, it is not surprising that PAQ-based compressors are currently among the best know general purpose compression techniques. They typically employ a large number of base predictors, some of which are specialized for certain types of data. In principle, predictive models like the SM could be integrated into context mixing compressors as one of the base predictors (some of them currently employ a variant of PPM). However, compression ratios of PAQ-based compressors are already very good and computational and memory requirements are currently the main reasons preventing their widespread adoption.

Since the original proposal [Gasthaus et al., 2010] of using the SM as a predictive model in the data compression setting, and based on the improvement presented in [Gasthaus and Teh, 2010], further refinements to this approach have been proposed [Bartlett et al., 2010; Bartlett and Wood, 2011]. These papers propose a method for turning the SM into a streaming predictor with finite, fixed memory footprint and linear computational complexity by dynamically pruning unused contexts from the context tree. This allows the resulting compressor to be applied to large files without the need to split the file into independent blocks, as is commonly done with other compressors. The authors experimentally demonstrate that by using a simple pruning strategy it is possible to significantly reduce the amount of memory required without deteriorating compression performance relative to the unconstrained method, while significantly outperforming the naïve block-wise approach.

6.2 (DE-)PLUMP

Our compression algorithm, dubbed "DEPLUMP"[2], draws on the improvements presented in the previous sections to meet the requirements of an online prediction algorithm that can be coupled to an adaptive entropy coder in order to yield a compression algorithm. In particular, we combine the online context tree construction algorithm (Section 5.1) and the compact representation (Section 4.3) with one of the sequential approximate inference schemes UKN (Section 5.2), 1PF (Section 5.3.4), or FRAC (Section 5.4) and stochastic gradient descent-based hyperparameter optimization (Section 5.7.2).

In order for a predictive model to be usable with arithmetic or range coding, it needs to able to deterministically make a prediction (in form of a CDF) for the next symbol and then update its state based on the observed symbol. The encoder and decoder need to use exactly the same probabilistic model, which is updated on the decoder side with the symbols that have already been decoded. In order for the encoder and decoder to use exactly the same model when Monte Carlo-based inference methods are used, the seeds of the pseudo random number generators used during sampling must either be fixed in advance or be included in the header of the compressed file.

As the details of the model and the online inference procedure have already been discussed in detail in previous sections, we will only give a pragmatic description of the resulting algorithm here, serving as a guide to implementors. Further, as implementing an adaptive entropy coder comes with its own algorithmic challenges but suitable library implementations are readily available, we will not discuss this part of the final (de-)compressor further.[3] At a high level, the overall (de-)compression algorithm proceeds as follows for each position $t = 1, \ldots, T$:

1. insert the current context $\mathbf{u} = x_{1:t-1}$ into the context tree (see Section 5.1); this may involve splitting an existing node and updating the representations of the resulting nodes using fragmentation (Section 5.6)

2. compute the predictive distribution $P_t(s \mid \mathbf{u})$ for all $s \in \Sigma$ using (2.45)

3. use the entropy coder to encode/decode x_t using P_t

[2] PLUMP is an acronym for Power-Law Unbounded Markov Predictor.

[3] The main difficulty in connecting the predictive model to the coder lies in the fact that arithmetic coding implementations (such as the ones described in [Witten et al., 1987; Nelson, 2014]) typically offer an interface where the predictive CDF has to be specified in terms of integer ranges, not in terms of floating point numbers (as typically output by probabilistic models), and a bit of care has to be taken to perform this quantization accurately without generating empty ranges. This is less of an issue when a 64-bit implementation of arithmetic coding is used.

4. Optionally: update the concentration and discount parameters using a gradient step to optimize $\log P_t(x_t \mid \mathbf{u})$

5. update the representation of the posterior distribution in light of the observation x_t using either one of the sequential approximate inference strategies described in Chapter 5; this involves updating the representations for some distance along the path from $\mathbf{u}$ to the root of the context tree

6. append x_t to the input buffer

In practice, all of the above steps can be interleaved into a single down-up traversal from the root of the tree to the insertion point of the context $\mathbf{u}$ and back up to the root. The computational complexity of a single insert-predict-update step is $O(\delta|\Sigma|)$ where $\delta = |(\varepsilon \rightsquigarrow \mathbf{u})|$ is the length of the path from the root to $\mathbf{u}$ in the context tree. In the worst case the tree grows as a linear chain, so that the overall complexity is quadratic in T. In practice on typical inputs however, the path lengths are on the order of $\log T$ for an overall complexity of $|\Sigma| T \log T$. As has been pointed out (and addressed) by Bartlett and Wood [2011], depending on how the fragmentation operation is implemented, its computational complexity can also grow linearly with T. In practice however, the growth is typically sub-linear in T and the predictive performance of an approximate, constant time fragmentation (see Section 5.6) is typically almost indistinguishable.

6.3 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* $\ell(x_{1:T}) = -\frac{1}{T}\sum_{t=1}^{T} \log_2 p(x_t|x_{1:t-1})$ 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

		DEPLUMP			PPM		CTW / CTS	
File	Size	1PF	UKN	FRAC	PPM*	PPMZ	CTW*	S-CTS
bib	111261	1.73	1.72	**1.71**	1.91	1.74	1.83	1.75
book1	768771	2.17	2.20	**2.14**	2.40	2.21	2.18	2.20
book2	610856	1.83	1.84	**1.80**	2.02	1.87	1.89	1.89
geo	102400	4.40	4.40	4.42	4.83	4.64	4.53	**3.60**
news	377109	2.20	2.20	**2.17**	2.42	2.24	2.35	2.34
obj1	21504	3.64	3.65	3.67	4.00	3.66	3.72	**3.40**
obj2	246814	2.21	**2.19**	2.20	2.43	2.23	2.40	**2.19**
paper1	53161	2.21	2.20	**2.19**	2.37	2.22	2.29	2.26
paper2	82199	2.18	2.18	**2.16**	2.36	2.21	2.23	2.22
pic	513216	0.72	0.72	**0.71**	0.85	0.76	0.80	0.76
progc	39611	2.23	**2.21**	**2.21**	2.40	2.25	2.33	2.30
progl	71646	1.44	1.43	**1.42**	1.67	1.46	1.65	1.59
progp	49379	1.44	**1.42**	1.43	1.62	1.47	1.68	1.61
trans	93695	1.21	**1.20**	1.21	1.45	1.23	1.44	1.35
avg.		2.12	2.12	**2.10**	2.34	2.16	2.24	**2.10**
w. avg.		1.89	1.91	**1.87**	2.09	1.93	1.99	**1.87**

Table 6.1: 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] and the results for Skip Context Tree Switching from [Bellemare et al., 2014].

of bytes, i.e. $|\Sigma| = 256$.[4]

The results are shown in Table 6.1. For comparison, we also show the results of two PPM variants, CTW* [Willems, 1998], and S-CTS [Bellemare et al., 2014] in the final four columns. PPM* [Cleary and Teahan, 1997] was the first PPM variant to use unbounded-length context, and the results for PPM-Z [Peltola and Tarhio, 2002] 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

[4] 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 $d_\infty = 0.95$ and the concentration parameter was fixed at $\alpha = 0$, i.e. the same hyperparameters also used in the language modelling setting (Section 5.7.2). The hyperparameters were optimized by online gradient ascent in the log-predictive probability with a learning rate of 10^{-4}, interleaved with the count updates. The initial hyperparameter values were chosen through manual experimentation on the Brown corpus validation set. The `news` file from the Calgary corpus was used for parameter selection in the compression setting (e.g. for choosing the learning rate), but no extensive hyperparameter tuning was performed.

results for UKN, 1PF, and FRAC are comparable, with no consistent advantage for any single approach on all files, though FRAC appears to perform slightly better on text files. FRAC thus provides a good trade-off between computational complexity and compression performance, while UKN can be chosen when memory usage and floating point operations are to be minimized. The comparison to S-CTS is interesting in that both achieve the same average compression ratio, but have very different strengths: while DEPLUMP performs particularly well on text data (`book*`, `news`, `paper*`), S-CTS achieves very good performance on binary (`obj*`) and structured data `geo`).

In addition to these experiments, we performed experiments with several variants of the basic algorithm: 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. Three main observations could be made from these experiments: 1) neither using independent discount parameters nor using more particles improves performance consistently; 2) using 100 particles yields ≈ 0.02 bps improvement on the large text files (`book`, and `paper`), but no improvement on the other files; 3) online gradient updates to the discount parameters consistently improve performance by ≈ 0.02 bps. Further, choosing a non-zero concentration parameter does not provide a significant performance improvement (even when also updated online), though it does help to alleviate the "runs problem" (see below).

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 challenge [Hutter, 2006]. On the Canterbury corpus, the SM-based compressor consistently outperformed the methods compared here, with the exception of two binary files.

On the Wikipedia excerpt, the UKN algorithm 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 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).

6.4 Practical Details And Improvements

One complication that arises when the SM is used for compression is that binary data files sometimes contain sequences that have very low probability under the HPYP model, leading to large losses when they occur. In particular, long runs of the same symbol are problematic in two ways: On the one hand they lead to worst-case computational complexity, on the other hand the probability of any other symbol occurring after a run goes to 0 quickly. In [Gasthaus et al., 2010] we proposed to deal with the second aspect of the problem by predicting the next symbol using a mixture of simple unigram model and the SM, where the unigram component has a very small weight. This mitigates the problem by ensuring that numerically the predictive probabilities never become exactly 0, but causes a slight degradation in performance for data that does not suffer from this problem.

An alternative solution is to apply run-length encoding (RLE) to the input before feeding it to the compressor, which is common practice for many other compression algorithms. This has the added benefit of removing the pathological cases that lead to worst case computational complexity. However, in order not to unnecessarily deteriorate the performance on files that do not contain many long runs, RLE should either be used adaptively (and its use signalled by a flag in the header), or should only be applied to runs exceeding some minimal length (e.g. 10). While performing RLE as a pre/post processing step is simple and effective in practice, performance can likely be improved even further by using a separate model to encode the run length information, instead of treating them as normal symbols in the SM model.

6.5 Discussion / Future Work

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/S-CTS. The reasons for this include both the fact that DEPLUMP uses unbounded context lengths and the fact that DEPLUMP employs an underlying probabilis-

tic 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].

The fact the DEPLUMP is based on a well-defined probabilistic model opens interesting avenues for further improvements to the compression performance. First, we note that pre-training of the predictive model can be desireable to do, particularly as compression of the first part of any file suffers as a result of the initially inaccurate model. Doing this kind of pre-training is known to help PPM compression (at the cost of having to transmit either the pre-trained model parameters or the files which were used for pre-training in addition to the file itself). In DEPLUMP, we could build on the domain adaptation work of Wood and Teh [2009] to introduce pre-training and model sharing in a principled way.

Additionally, both PPM and DEPLUMP assume (implicitly in the case of PPM, explicitly in the case of DEPLUMP) that the underlying symbol generation distribution is stationary.[5] Extensions to nonparametric Bayesian models that account for non-stationarity such as those described in the dependent Dirichlet process literature [Caron et al., 2007; MacEachern, 2000] can also be applied here. Intuitively, doing this will allow the compressor's prediction model to adapt to the statistics local to a particular region of a file.

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. 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.

For CTW/CTS where the marginalization over latent variables can efficiently

[5] Strictly speaking, this is only true for DEPLUMP if the concentration and discount parameters are kept fixed and not updated online.

be performed analytically, it has been possible to establish theoretical guarantees of performance in the form of regret bounds and proofs of asymptotic optimality wrt. the type of source considered [Willems et al., 1995; Veness et al., 2012; Bellemare et al., 2014]. Deriving similar guarantees for the SM and DEPLUMP, an endeavour that is complicated by the use of approximate inference, is an interesting avenue for future work.

As a final thought, it is interesting to note that hierarchical PYP models were originally intended for modelling sequences of words coming from a large (even infinite) vocabulary [Teh, 2006a,b]. Here we have used essentially 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 points to an interesting further application of DEPLUMP, namely to large alphabet texts, e.g. of Chinese and Japanese.

6.6 Sequence Memoizer Software: `libplump`

The Sequence Memoizer model, including all described CRP representations, as well as all presented online and offline inference algorithms, is implemented in the freely available `libPLUMP` software package.[6]

`libPLUMP` is mostly written in C++ for efficiency and to allow for tight control of memory usage, allowing it to handle models for tens to hundreds of millions of tokens on typical hardware. In the online setting it an make predictions and update the model at speeds exceeding 50000 symbols/second, depending on the data set and parameter settings. As `libPLUMP` is primarily a research tool, it is designed to make it easy to extend, e.g. with different CRP representations, online and offline inference algorithms, as well as hyperparameter optimization schemes. The modular C++ structure is exposed via Python bindings, to allow for easy scripting and interactive exploration. The majority of experiments described in this thesis were performed using this library (or earlier versions of it).

The following is a simple illustration how the Python interface can be used to build a Sequence Memoizer model and make predictions.

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

```
import libplump

restaurant = libplump.HistogramRestaurant()

node_manager = libplump.SimpleNodeManager(restaurant.getFactory())
parameters = libplump.SimpleParameters()

seq = libplump.VectorInt([0,1,2,1,2,1,1])
N = len(seq) - 2 # training length
num_types = 3

model = libplump.HPYPModel(seq, node_manager, restaurant,
                           parameters, num_types)

# online inference / prediction
training_losses =  model.computeLosses(0, N)

# sampling
for iter in range(10):
  model.runGibbsSampler()

# prediction
for i in range(N, N + 2):
   dist = model.predictiveDistribution(N, i)
   print dist, sum(dist)

# save model to file
serializer = libplump.Serializer("model.dump")
serializer.saveNodesAndPayloads(nodeManager, restaurant.getFactory())
```

CHAPTER 7

Sequence Memoizer Cache Language Models

7.1 Introduction

The models described previously assume that the underlying generative process is *stationary*, i.e. doesn't change with "time" (positions in the input). For language modelling, where the input is usually a concatenation of different documents, and documents consist of multiple sections and paragraphs, this assumption does generally not hold [Johansson, 1985]. The vocabulary used in different documents or different sections within a document can change considerably (consider e.g. a sports news article vs. an article on politics), as can the frequency with which particular syntactic constructions are used. The term *burstiness* is typically used to refer to a particular instance of this phenomenon, namely the observation that if a term appears in a document at all, the probability that it appears again is drastically increased (over the relative frequency of the term in the entire corpus) [Katz, 1996; Church and Gale, 1995]. This phenomenon is mainly observed for *content words* (e.g. nouns, verbs, adjectives), and less so for *function words*, leading some authors to include part-of-speech information into their models [Kuhn and De Mori, 1990]. Content words exhibit both *document-level* as well as within-document burstiness, where, according to the definitions of Katz [1996], the former refers to the fact that most content words appear only in few documents, and the latter to the observation that even within documents occurrences of the same content word tend to cluster together.

A similar phenomenon can also be observed in data compression, where binary files often consist of different sections (e.g. headers and different types

of content blocks) that have different marginal and conditional symbol distributions. Many compression techniques therefore employ techniques for adapting to non-stationary input (e.g. by breaking up the input into fixed-size blocks, or by using other techniques, such as the count-halving employed by variants of PAQ) [Mahoney, 2013].

Not surprisingly, there have been numerous attempts to integrate models of burstiness into probabilistic language models, ranging from complex models incorporating rich domain knowledge, e.g. in the form of topic or dialog models [Iyer and Ostendorf, 1996; Clarkson and Robinson, 1997; Lucas-Cuesta et al., 2013], to simple (yet effective) *cache* language models, which we will consider here.

More broadly, adapting a model to the local context can be seen as an instance of *domain adaptation*, i.e. using training data that is similar to the one in the target domain to improve model performance in the target domain, which has also been explored in the context of statistical language models. In particular, the possibility of adapting a statistical language model trained on a (large) corpus of text to a new domain with different characteristics (where less data is available) has been extensively studied [Kneser and Steinbiss, 1993; Bacchiani and Roark, 2003; Bellegarda, 2004; Wood and Teh, 2009; Alumäe and Kurimo, 2010; Grave et al., 2016].

Here, we will consider how non-stationarity in general and burstiness in particular can be modeled in the context of PYP-based language models. We will use the idea of a local "cache" model, i.e. a model trained on only a small part of the training corpus preceding the prediction location. To construct such a cache model in the HPYP setting, we make use of ideas from work on time-varying Dirichlet/Pitman-Yor processes by Caron et al. [2007] and Bartlett et al. [2010].

We will first describe related work on non-stationary language models and time-varying HPYP models, then describe our approach, empirically evaluate the models' predictive performance, and conclude with an outlook on future work.

7.2 Background

7.2.1 Cache Language Models

A simple yet effective approach to modelling non-stationarity are so-called *cache language models* [Kuhn, 1988; Kuhn and De Mori, 1990; Jelinek et al.,

1991; Clarkson and Robinson, 1997; Goodman, 2001b]. These models estimate a "local" model (typically an n-gram model) from the current context (e.g. the current paragraph, the current document, or a sliding window preceding the current position), which is then blended (typically linearly) with another, stationary base language model. The blending itself can be *dynamic*, i.e. dependent on the local context [Kneser and Steinbiss, 1993]. The result of using such a cache model in combination with a stationary language model is typically a significant reduction in perplexity, leading Goodman [2001b] to the conclusion that "[...] caching is potentially one of the most powerful techniques we can apply [...]".[1] Jelinek et al. [1991] report a 8% to 23% reduction in perplexity when linearly combining a stationary trigram model with a trigram cache using the most recent 1000 words, and Goodman [2001b] reports significant improvements (up to 0.6 bits/symbol) for combining a smoothed trigram baseline model with a unigram, bigram, or trigram cache (with a significant performance improvement between unigram and bigram cache, and a small improvement when going from bigram to trigram cache). More recently, Mikolov et al. [2011] report a $\approx 10\%$ drop in perplexity when combining a unigram cache with a 5-gram model on the PTB corpus.

Part of these improvements is due to the fact that the cache is updated on-line while making predictions (on the test set), thereby increasing the total amount of training data used, whereas non-cache stationary models are usually kept fixed after training. Thus, part of the improvement can be achieved without the cache component simply by updating the "stationary" model with new training data as it is observed, e.g. using the online inference procedure described in Chapter 5 (cf. the "online" column in Table 5.1). This idea of "dynamic evaluation" has also been explored in the context of neural language models [Mikolov, 2012; Krause et al., 2017, 2019]. However, depending on the model and the application, updating the base model online may be computationally infeasible (e.g. in mobile text entry applications).

More broadly, a *cache language model* is a language model that incorporates information about the local data statistics by estimating a separate model based on a buffer (cache) of the most recently observed symbols, and then combining this model with a globally-estimated base model. In the usual setup, the base model is trained and optimized on a training sequence $\mathbf{x}^{\text{train}}$, and predictions need to be made incrementally symbol-by-symbol for a new sequence $\mathbf{x}^{\text{test}}$.

[1] However, Goodman [2001b] also cautions that when caching is used in applications such as speech recognition, where the input to the cache model is not the ground truth but the output of a system that makes errors, achieving good performance in practice might be much harder.

In the most commonly-used variant, the combined model is simply a mixture model with two components: the base model component $P^{\text{base}}(\cdot)$ estimated from the training sequence and then fixed, and a local (cache) model component $P_i^{\text{cache}}(\cdot, x_{i-L:i-1})$ estimated from the L most recent symbols of the (test) input $\mathbf{x}$, i.e.:

$$P_i(s) = \lambda P^{\text{base}}(s) + (1-\lambda) P^{\text{cache}}(s, x_{i-L:i-1}), \tag{7.1}$$

where the forms of both P^{base} and P^{cache} can be chosen arbitrarily. In practice, the model chosen for the cache component is typically simple, as it needs to be incrementally updated (or re-estimated) for each symbol, making n-gram models estimated using the MLE (2.8) a common choice. Cache models where the probability of a word under the cache component decays exponentially with the distance of previous occurrences of the word have also been considered [Clarkson and Robinson, 1997; Tiedemann, 2010].

More recently, the effectiveness of cache models has also been considered in the context of neural language models. Mikolov [2012] demonstrated that including a simple unigram cache can significantly improve performance of ensembles of RNN language models and n-gram models. Grave et al. [2016] propose a "Neural Cache Model" where the cache is integrated into the neural language model directly and acts not on the observed sequence but on the hidden model activations. This architecture achieves significant reductions in perplexity relative to the non-cache baseline mode, from 82.3 to 72.1 on the Penn Tree Bank data set. This work has further been extended in [Grave et al., 2017] to incorporate larger context.

7.2.2 Time-Varying HPYP Models

The work most closely related to the model we are proposing here is the work by Bartlett et al. [2010], who develop a time-varying HPYP model and associated approximate inference procedure to enable "constant-space" inference in the Sequence Memoizer model. However, the goal of [Bartlett et al., 2010; Bartlett and Wood, 2011] is not adaptation to a non-stationary data stream, but to limit the overall size of a Sequence Memoizer model by removing customers and entire restaurants from the context tree and associated Chinese restaurant franchise representation as data is processed in streaming fashion.

The key idea in their approach is that instead of assuming a single context-tree-structured set of distributions $\{G_{\mathbf{u}}\}$ from which all observations $x_{1:T}$ are drawn, the set of distributions is taken to be time-varying, i.e. there is a set of distributions $\{G_{\mathbf{u}}^t\}$ for each $t = 1, \ldots, T$. These distributions are assumed to be

dependent through time, i.e. each $G^t_{\mathbf{u}}$ depends on $G^{t-1}_{\mathbf{u}}$, allowing estimation even when only a single observation is made per time step.

The underlying idea of constructing dependent distributions based on the Dirichlet or Pitman-Yor process has been extensively explored in the literature and has been successfully brought to bear on applications using time-varying mixture and admixture models [MacEachern, 1999, 2000; Srebro and Roweis, 2005; Griffin and Steel, 2006; Griffin, 2007; Caron et al., 2007; Gasthaus et al., 2009; Sudderth and Jordan, 2009; Rao and Teh, 2009]. While many such constructions are based on inducing dependence in the stick-breaking representation, the particular construction used in [Bartlett et al., 2010], which is the one we will also make use of here, induces a dependence between two PYP-distributed random variables in the CRP representation.

This construction, originally proposed for Dirichlet process mixture models in [Caron et al., 2007], exploits a fundamental consistency property of the CRP that ensures that restricting a given CRP partition $A \sim \mathrm{CRP}_c(\alpha, d)$ to any subset $B \subset [c]$ chosen independently of A (and relabeling the customers) yields another CRP-distributed partition [Pitman, 2002]. In particular, the construction employed by Caron et al. [2007] and Bartlett et al. [2010] relies on *deleting* (i.e. removing) customers from a CRP partition for $G^{t-1}_{\mathbf{u}}$ to construct a CRP partition for $G^t_{\mathbf{u}}$.

In particular, sampling $A_1 \sim \mathrm{CRP}_{n_1}(\alpha, d)$, constructing a partition B by removing $l \leq n_1$ customers uniformly at random and relabeling the remaining customers $1, \ldots, n_1 - l$, yields a partition $B \sim \mathrm{CRP}_{n_1-l}(\alpha, d)$ that is marginally also distributed according to the CRP [Pitman, 2002; Caron et al., 2007; Bartlett et al., 2010]. The resulting partition B can then be extended by seating an additional n_2 customers through the sequential CRP (2.24) to a partition $C \sim \mathrm{CRP}_{n_1-l+n_2}$. Finally, this partition C can then be restricted to a partition A_2 containing only the final n_2 customers, yielding, after relabeling, $A_2 \sim \mathrm{CRP}_{n_2}(\alpha, d)$. This construction yields partitions A_1 and A_2 that are marginally distributed according to the CRP, but that exhibit a dependence through the non-deleted customers. The induced dependence structure is complex, but the degree of dependence depends on the number of customers l that are deleted, with stronger dependence the fewer customers are deleted.

Caron et al. [2007] uses this construction to induce dependence in a Dirichlet process mixture model, while Bartlett et al. [2010] use a variant of the same mechanism to restrict the size of a Sequence Memoizer language model. In particular, the algorithm proposed in Bartlett et al. [2010] uses an extreme form of this deletion procedure, where all customers in a given leaf context $\mathbf{u}$

are removed when the model exceeds a pre-specified size threshold, thereby keeping the overall size of the model bounded. They compare two procedures for choosing the context $\mathbf{u}$ to remove: either uniformly at random from all leaf nodes, or using a greedy procedure that selects the context with the smallest impact on the data likelihood, with the greedy procedure performing marginally better in practice.

7.3 Model: Sequence Memoizer with Deletion (SM-del)

In order to model non-stationarity and burstiness in the HPYP/SM framework, we propose to combine an stationary Sequence Memoizer base model with a time-varying SM model that acts as a cache.

In the simple cache language model, the cache component is an n-gram model that models the statistics of the most recent L characters. One key feature of n-gram models that enables them to be used in this setting is that they can quickly be updated to account for the shifting data window by simply decrementing the counts associated with the token moving out of the window on the left and incrementing the count for the newly observed token.

Here, instead of using an n-gram cache, we propose to combine a Sequence Memoizer base model with a cache component that is itself a Sequence Memoizer model. In principle one could achieve this by re-estimating the cache SM model at every position t based on the preceding L tokens. However, re-estimating the model for every token is computationally infeasible in practice.

To make using a Sequence Memoizer model as cache practical, we need to not just be able to update the model incrementally for each new symbol (e.g. using the techniques described in Chapter 5), but also to update the model by removing the effect of observations that fall out of the local length-L cache window. We will refer to such a model that "deletes" the effect of observations as SM-del.

Recall from (2.46) that in order to evaluate the predictive distribution under the SM model one needs to compute an expectation under the posterior distribution over the sufficient statistics $\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}$ of the Chinese restaurant franchise. This expectation can be evaluated either using a deterministic approximation, e.g. the "unbounded Kneser-Ney" (UKN) approximation discussed in Section 5.2 or the "fractional counts" approximation (Section 5.4), or using a Monte Carlo approximation based on samples drawn from the posterior distribution, e.g. using Gibbs sampling (Section 4.5) or Sequential Monte Carlo (Section 5.3).

In order to make caching practical, we thus need to be able to update either

representation (deterministic or sample-based) of the posterior distribution given the previous L symbols, $P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}|x_{i-L:i-1})$, to a representation of the posterior distribution given the previous $L-1$ symbols, $P(\{c_{\mathbf{u}s}, t_{\mathbf{u}s}\}|x_{i-(L-1):i-1})$ (i.e. removing the effect of the first symbol) in the cache window. For the UKN approximation this can easily be accomplished: Recall that under the UKN approximation, $t_{\mathbf{u}s} = 1$ if $c_{\mathbf{u}s} \geq 1$, i.e. all customers of a given type s sit at the same table. In order to remove the customer corresponding to the observation x_{i-L}, we locate the context for this observation, i.e. $\mathbf{u} = x_{1:i-L-1}$, in the context tree. We can then decrement the count $c_{\mathbf{u}s}$, and, if this results in $c_{\mathbf{u}s} = 0$, we decrement $t_{\mathbf{u}s}$ as well. If we decremented $t_{\mathbf{u}s}$, we also decrement $c_{\pi(\mathbf{u})s}$, and potentially adjust $t_{\pi(\mathbf{u})s}$, and so on, recursively, thus exactly reversing the process of inserting the customer. However, in the (Sequential) Monte Carlo setting, it is not obvious how to remove the effect of an observation x_{i-L} from a sample from the posterior given $x_{i-L:i-1}$.

Instead of trying to perform approximate inference inference in an SM model given only last $L-1$ symbol, we can achieve the desired effect of a locally adapting model by making use of the "forgetting counts" scheme for time-varying PYP models described above [Bartlett et al., 2010]. In order to remove the effect of the symbol x_{i-L} we perform a stochastic recursive deletion procedure like the one used for the SM Gibbs sampler. In particular, given a CRP representation of the data up to symbol i obtained by the SMC inference procedure, we can remove a customer of type x_{i-L} from the restaurant corresponding to context $x_{1:i-L-1}$ (by calling REMOVECUSTOMER($x_{1:i-L-1}$, x_{i-l}) from 2) before updating the representation to include the new observation x_i.[2]

Concretely, we propose interleaving the online SMC inference procedure described in Chapter 5 (Algorithm 3) with the stochastic REMOVECUSTOMER operation, performed before a new customer is inserted into the particle. If multiple particles are used, the deletion is performed for each particle independently. In the experiments we will refer only to the one-particle variant (1PF) unless otherwise noted.

We can either interpret applying this deletion step as a stochastic approximation for removing the effect of the symbol falling out of the cache window, or as an instance of the aforementioned "forgetting counts" time-varying HPYP

[2]Note that while based on the same underlying idea, this is different from the deletion procedure proposed in Bartlett et al. [2010]: In their work, the goal is to reduce the model size and they achieve this by deleting entire restaurants/contexts which are chosen either randomly or based on a greedy heuristic. By doing so they are able to achieve predictive performance similar to the original model, but with a model that does not grow without bound. Here, we want to achieve the opposite: we want the model to adapt to the *local* statistics of the data, reflecting our assumption that the underlying generative process is not stationary.

model. characters. The resulting SM-del model can then be combined in a mixture model (cf. Section 2.2.2) with an SM model trained on the full data (which can optionally be updated online as well).

7.4 Experiments

In order to evaluate the effectiveness of the proposed SM-del model, we consider multiple ways of combining the SM-del model with Sequence Memoizer base models and other n-gram cache models.

7.4.1 Cache Size & Mixture Weight

In a first set of experiments, we explore the effect of adding a lag-L SM-del cache component to an SM base model (either fixed or updated online), both of which are inferred independently and their predictions combined in a convex combination with fixed mixture weight λ. The results for this experiment on the Brown corpus are summarized in Figure 7.1.

Several observations can be made here: Without any cache, the gap between an offline Sequence Memoizer model that is not updated while making predictions on the test set, and a model that is updated online, is about 0.3 bits/symbol (8.07 vs. 7.77 bits per symbol). As expected, the offline model benefits more from the inclusion of the cache component, so that even for small mixing weights the gap between the online and offline model drops to about 0.1 bits/symbol, but doesn't disappear completely, suggesting that the SM-del cache component captures local effects that the base model cannot adapt to. Further, the optimal mixture weight of the cache component is slightly larger for the offline model.

The optimal cache window size L is around 1500 symbols for both models, but the offline model benefits more from being combined with a longer-lag cache. The effect of the interaction of mixture weight and window size is shown in Figure 7.2 for the offline model. As expected, the optimal cache window size increases slightly with the mixture weight. The optimal mixture weight dominates over a wide range of cache sizes.

7.4.2 Predictive Performance

We evaluated the predictive performance of linear combinations of three Sequence Memoizer base models (offline, online, and online with deletion) with SM-del cache components and n-gram cache components on the Brown cor-

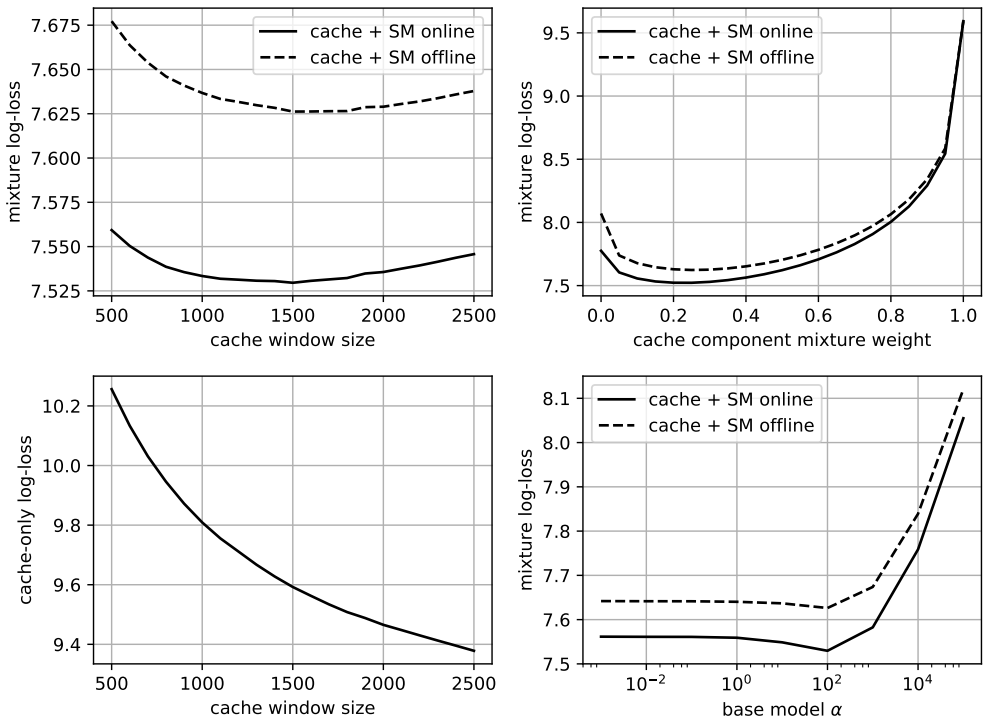


Figure 7.1: Predictive performance of a mixture model consisting of an SM base model combined with an SM-del cache components on the Brown corpus test set. The top left shows the test set log-loss as a function of the cache window size L. The best performance is obtained for a window size of $L = 1500$. The top right also shows test set log-loss, but as a function of mixture weight λ of the cache component (with $L = 1500$). The left and right boundaries show the performance of the base model and the cache models in isolation, respectively, with the optimum lying at 0.3. The bottom left shows the performance of the cache component in isolation, as a function of the window size. The bottom right shows the performance of the mixture as a function of the concentration parameter α used for the base model (with $L = 1500$ and $\lambda = 0.3$). The optimum is at $\alpha = 100$, which is used for the other plots. The discount parameters for all models are the "standard" values described in Section 5.7.2, and the models are constructed using the 1PF algorithm.

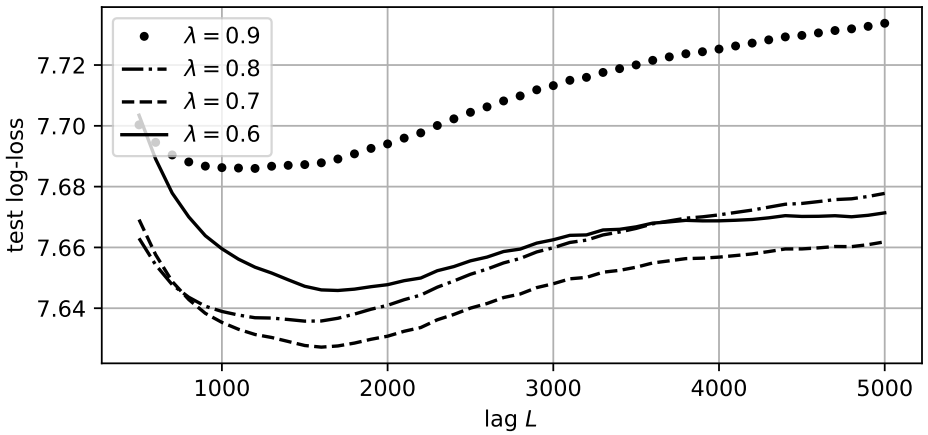


Figure 7.2: Predictive performance (test set log-loss) of an SM base model with an SM-del cache as a function of cache window size L for several mixture weights λ.

Model	**Brown**	**Penn Treebank**
Sequence Memoizer (offline)	8.07	7.13
+ 1-gram cache $L = 1500$	7.79	7.04
+ 3-gram cache $L = 1500$	7.93	7.09
+ 1,2,3-gram cache $L = 1500$	7.67	6.97
+ 1-gram cache $L = 10^2, 10^3, 10^4$	7.75	6.97
+ 3-gram cache $L = 10^2, 10^3, 10^4$	7.90	7.08
+ 1,2,3-gram cache $L = 10^2, 10^3, 10^4$	7.61	6.90
+ SM-del $L = 1500$	7.63	6.96
+ SM-del $L = 10^2, 10^3, 10^4$	**7.57**	6.91
+ SM-del + 1,2,3-gram cache $L = 10^2, 10^3, 10^4$	**7.57**	**6.89**
Sequence Memoizer (online)	7.77	7.01
+ 1-gram cache $L = 1500$	7.62	6.94
+ 3-gram cache $L = 1500$	7.74	7.00
+ 1,2,3-gram cache $L = 1500$	7.56	6.91
+ 1-gram cache $L = 10^2, 10^3, 10^4$	7.59	6.88
+ 3-gram cache $L = 10^2, 10^3, 10^4$	7.73	7.00
+ 1,2,3-gram cache $L = 10^2, 10^3, 10^4$	7.52	6.84
+ SM-del $L = 1500$	7.53	6.90
+ SM-del $L = 10^2, 10^3, 10^4$	7.49	6.85
+ SM-del + 1,2,3-gram cache $L = 10^2, 10^3, 10^4$	**7.48**	**6.83**
Sequence Memoizer (online/w del. $L = 1500$)	7.89	7.03
+ 1-gram cache $L = 1500$	7.73	6.96
+ 3-gram cache $L = 1500$	7.85	7.03
+ 1,2,3-gram cache $L = 1500$	7.65	6.93
+ 1-gram cache $L = 10^2, 10^3, 10^4$	7.69	6.90
+ 3-gram cache $L = 10^2, 10^3, 10^4$	7.84	7.02
+ 1,2,3-gram cache $L = 10^2, 10^3, 10^4$	7.59	6.86
+ SM-del $L = 1500$	7.62	6.92
+ SM-del $L = 10^2, 10^3, 10^4$	7.56	6.88
+ SM-del + 1,2,3-gram cache $L = 10^2, 10^3, 10^4$	**7.55**	**6.86**

Table 7.1: Predictive performance comparison of a Sequence Memoizer model when compared with different forms of cache models: a unigram and a trigram cache, and the proposed SM cache model. The top set of rows uses a base SM model that is estimated on the training set and then fixed, while the bottom set of rows updates the model online on the test set. For this comparison the hyperparameters were set at $\lambda = 0.3$, $L = 1500$, and $\alpha = 100$, as well as the "standard"' discount parameters.

pus and the Penn Treebank dataset (see Appendix B). The predictive performance of the various model combinations in terms of log-loss on the test set are summarized in Table 7.1. The underlying base Sequence Memoizer model discount parameters were kept fixed to the "standard" setting (cf. Sec. 5.7.2), $d_{0:9} = (0.05, 0.7, 0.8, 0.82, 0.84, 0.88, 0.91, 0.92, 0.93, 0.94)$, $d_\infty = 0.95$. The concentration parameter $\alpha = 100$ and the window size $L = 1500$ were chosen based on the experiments shown in Figure 7.1, where the effect of these parameters was individually evaluated on the Brown corpus test set. To validate the performance of the resulting model on completely unseen test data, the second column in Table 7.1 shows the performance using the same hyperparameters on the Penn Treebank dataset (i.e. no hyperparameter tuning was performed on that dataset). For all experiments, the offline base model was estimated on the train+validation set and then fixed; the online variant was incrementally updated while making predictions; the "online with deletion" variant was estimated without deletion on the train+validation set, and then updated while making predictions on the test set while also performing deletion at a lag of $L = 1500$. For each combination, the mixing weights were optimized on the respective validation set by first training the base model on the training set and making predictions on the validation set. The mixing weights were then fixed, the validation set folded into the training set, and the base model re-estimated on the combined train+validation set.

Overall, it can be seen that including the proposed SM-del cache component leads to improved performance over including simpler uni- or trigram cache models, both in the setting where the base model is kept fixed and when it is updated online. For both datasets, the same pattern in terms of performance emerges:

- The SM model that is updated online outperforms the fixed model independently from the cache component that is added;
- Adding any form of caching improves the model's predictive performance;
- When only a single lag ($L = 1500$) is considered, adding an SM-del component outperforms a unigram, trigram, and a combined uni-/bi-/trigram (denoted 1,2,3-gram) cache component;
- Combining multiple lags, as e.g. proposed by Mikolov [2012] as a form of cache decay, further improves performance of both n-gram caches and the SM-del model.

- Combining multiple n-gram and SM-del models at different three different lags with an SM base model that is updated online (for a total ensemble with 13 models) yields the best performance.

Interestingly, combining the three different SM base models (offline, online, online with deletion) does not yield additional improvements over just the online model that performs best.

For comparison, on the Penn Tree Bank data set, Mikolov [2012] reports a perplexity of 141.2 (7.14 bps) for a modified Kneser-Ney smoothed 5-gram model, which is reduced to a perplexity of 125.7 (= 6.97 bps) when the same model is interpolated with a collection unigram cache models with different lags (the exact number and values of the lags used a not further specified).

Goodman [2001a] reports that using an interpolated trigram cache model (interpolated with uni- and bi-grams) leads to almost twice the percentage improvement then using a unigram cache. We observe a similar improvement, yet somewhat less pronounced effect: comparing the difference between the "1-gram cache", the "1,2,3-gram cache", and the SM-del rows (in our notation "3-gram" refers to an un-interpolated trigram model, whereas the "1,2,3-gram" is the interpolated version as used by Goodman [2001a]).

7.5 Discussion

We have proposed a cache language model variant of the Sequence Memoizer, SM-del, based on incorporating "deletion" into the online SM inference procedure. The straight-forward modification of the approximate inference procedure, which makes use of the REMOVECUSTOMER sub-routine also used in the Gibbs sampling schemes, can alternatively be interpreted as a time-varying variant of the hierarchical PYP model, proposed in similar form in previous work [Bartlett et al., 2010; Caron et al., 2007]. While the modification is simple, it proves to be quite effective in boosting the predictive accuracy when linearly combined in an ensemble with a base SM model that is updated online. As we have demonstrated empirically, the proposed SM-del cache component compares favorably in terms of predictive performance to n-gram cache models.

For future work, there are two direct extensions of the work presented here that can be explored: In the experiments presented above, we have only evaluated the model performance using the (modified) one-particle particle filter inference scheme, which is a somewhat crude (albeit effective) heuristic. One open question is thus how the SM-del model performance varies when dif-

ferent inference procedures are used. An MCMC sampling approach in the time-varying model could be considered, though it may come with a computational cost that is prohibitive for practical applications. Initial experiments with using larger number of particles did not yield significant improvements. Deletion schemes for the FRAC approximations could also be considered.

Another direction for future work is exploring alternative ways for combining the cache model with the base model. Here, we have only considered simple linear combinations. However, more sophisticated variants, e.g. using the Graphical Pitman-Yor framework (see Section 8.1.1) could be considered. Initial experiments in this direction have been promising and some model variants are implemented in our ScaNPB library (Sec. 8.7)). However, these models are considerably more complex than the proposed simple scheme, which may already achieve a large fraction of the possible gains.

CHAPTER 8

Hybrid PYP-based and Neural Language Models

This chapter describes initial work on several hybrid models that combine count-based PYP language models with a simple neural language model, the log-bilinear model. While the resulting models and the associated inference procedure are more complex, the predictive performance does not (yet) exceed that of simpler techniques, and further in-depth study of models of this type is needed, we include this work here as a starting point in a promising direction of future research.

8.1 Introduction

Neural language models [Miikkulainen and Dyer, 1991; Bengio et al., 2003; Mnih et al., 2009; Mikolov, 2012; Zaremba et al., 2014; Józefowicz et al., 2016; Dai et al., 2019; Radford et al., 2019], i.e. language models based on artificial neural networks, are an alternative class of models that rely on a different mechanism for overcoming the data sparsity problem. While count-based models, including the ones described in previous chapters, work by combining distributions from related contexts to produce smoothed estimates, neural language models induce real-valued *representations* of words, allowing them to base their probability estimates on learned semantic and syntactic similarities between the context word(s) and the predicted word.

Neural language models, in particular recent approaches based on recurrent neural networks (RNNs) [Mikolov, 2012; Zaremba et al., 2014] and attention-based transformer models [Vaswani et al., 2017; Radford et al., 2018, 2019; Dai et al., 2019], are among the best-performing language models known

to date. For some of these models (e.g. [Mnih et al., 2009; Mikolov, 2012]) their predictive performance can be significantly increased further by combining them with an n-gram language model after training, suggesting that these model classes have complementary strengths. The combination is typically performed using simple linear interpolation of the conditional probabilities using fixed weights, i.e.

$$\mathrm{P}(x_t|x_{1:t-1}) = \lambda \mathrm{P}_{n-\mathrm{gram}}(x_t|x_{1:t-1}) + (1-\lambda)\mathrm{P}_{\mathrm{neural}}(x_t|x_{1:t-1}), \qquad (8.1)$$

where the optimal mixing weight $0 \leq \lambda \leq 1$ is determined on a validation set [Mikolov, 2012].

This chapter explores "hybrid" models that combine a (simple) neural language model and a count-based model at a different, more fine-grained level. In particular, we explore several models that use the log-bilinear model (LBL) introduced by Mnih and Hinton [2007] as part of the base distribution in a hierarchical Pitman-Yor process language model. We choose the LBL here for its simplicity, as our focus is not on achieving state-of-the-art predictive performance, but rather to investigate the properties of such hybrid models qualitatively. In terms of pure predictive performance, state-of-the-art neural language models (e.g. [Zaremba et al., 2014; Józefowicz et al., 2016; Merity et al., 2017; Dai et al., 2019; Radford et al., 2019] can drastically outperform the models presented here.

Goldwater et al. [2011] argued that explicitly incorporating power-law properties into generative models (e.g. by using PYP priors) allows other parts of the models to work and discover structure more effectively, as they are not "burdened" with generating the power-law aspects of the data. In the previous chapters we have seen that this approach is effective for models of power-law sequences, and others [Blunsom and Cohn, 2011; Goldwater et al., 2011] have shown that it is helpful for discovering linguistic structure using latent variable models. The idea underlying the models presented here is that by using a PYP "adaptor" (to borrow the terminology of Goldwater et al. [2006b]) allows the neural language model to operate more effectively as it does not have to use its flexibility to model the power-law behavior of the token distributions.

We will first review the *graphical Pitman-Yor process* [Wood and Teh, 2009], which we will make use of as part of one of the explored hybrid models, and which by itself is an interesting extension to HPYP modelling framework. Subsequently we will describe the log-bilinear neural language model [Mnih and Hinton, 2007; Mnih and Teh, 2012], which is the neural language used in our models. We will then describe the hybrid models, the inference and learning procedure, followed by experimental results, analysis, and discussion.

Conceptually, our work is close to the work of Goldwater et al. [2006b] in using the PYP to model the power-law aspects of natural language text that might otherwise interfere with other aspects of the model when not modelled explicitly. In terms of models, we build upon the work of Wood and Teh [2009], who introduced the Graphical Pitman-Yor Process, and on the work of Mnih and Teh [2012] who introduced the log-bilinear language model. Our models are also related to the adaptive mixture model of Kneser and Steinbiss [1993], who propose to learn mixing weights for a fixed set of language models in an adaptive fashion, not unlike the way context mixing compressors like PAQ [Knoll, 2011] operate. More recently, Neubig and Dyer [2016] explored an alternative way of integrating count-based and neural language models more deeply by using the n-gram probabilities as additional features in a neural model, and obtained state-of-the-art performance.

8.1.1 The Graphical Pitman-Yor Process

In the hierarchical Pitman-Yor process models described in the earlier chapters, the base measure for a PYP-distributed random measure $G_{\mathbf{u}}$ that is part of a hierarchical prior, was either another PYP-distributed random measure $G_{\pi(\mathbf{u})}$ or a fixed, non-random measure (e.g. the uniform distribution at the top of the HPYP hierarchy). The *graphical Pitman-Yor process* (GPYP), originally proposed by Wood and Teh [2009] in the setting of language model domain adaption, extends this construction by allowing the base distribution to be a mixture of two (or more) random or non-random distributions, thus allowing the structure of the underlying hierarchy to be a directed acyclic graph (DAG), rather than a tree.

More formally, given some DAG $\mathcal{G} = (V, E)$ with vertex set V and edge set $E \subset V \times V$ which defines the hierarchical model structure, we associate a distribution G_v with each vertex $v \in V$, some subset $\tilde{V} \subset V$ of which are PYP-distributed random distributions,

$$G_u \sim \mathrm{PY}\left(\alpha_u, d_u, \sum_{v \in \pi(u)} \lambda_{v \to u} G_v\right) \qquad \forall u \in \tilde{V} \tag{8.2}$$

where $\tilde{V} \subset V$, $u, v \in V$ are vertices, $\pi(u)$ denotes the set of parents of u, $\pi(u) = \{v | (v,u) \in E\}$, and $\lambda_{v \to u}$ is a positive weight attached to the edge (v,u), with the constraint that the weights on all incoming edges into a node u sum to 1,

$$\sum_{v \in \pi(u)} \lambda_{v \to u} = 1. \tag{8.3}$$

In other words, the base distribution for each PYP-distributed random variable $v \in \tilde{V}$ is a mixture whose components are the parents $\pi(v)$ of v in the graph and the mixing weights are given by weights on the incoming edges.

8.1.2 CRP Representation of the GPYP

The GPYP can also be described in terms of a hierarchical CRP-based construction, dubbed the "multi-floor Chinese restaurant process" (m-CRP) by Wood and Teh [2009]. The main idea is that, just as in a simple finite mixture model, the mixture base distribution can equivalently be described as the marginal distribution of a process that associates an indicator latent variable to each draw from the base distribution, which selects one of the mixture components. For a simple finite mixture model $P(x) = \sum_{j=1}^{J} \lambda_j P_j(x)$ we can introduce an indicator variable z with distribution $P(z = j) = \lambda_j$, and write

$$P(x) = \sum_{j=1}^{J} P(x, z = j) = \sum_{j=1}^{J} P(z = j)P(x|z = j) = \sum_{j=1}^{J} \lambda_j P_j(x).$$

Recall that in the CRP construction of a PYP model, $G \sim \mathrm{PY}(\alpha, d, H)$, $x_{1:N} \overset{\text{iid.}}{\sim} G$, we have $(a_1, \ldots, a_K) \sim CRP_N(\alpha, d), \phi_{1:K} \overset{\text{iid.}}{\sim} H, x_i = \varphi(A, \phi_{1:K}, i)$ with quantities defined as in Section 2.3.2. Now consider the basic GPYP model

$$G \sim \mathrm{PY}\left(\alpha, d, \sum_{j=1}^{J} \lambda_j H_j\right) \qquad x_{1:N} \overset{\text{iid.}}{\sim} G. \tag{8.4}$$

The only difference between this model and the basic PYP model is that the base distribution is a mixture, so the "dishes" $\phi_{1:K}$ in the CRP construction are iid. draws from the mixture $\sum_{j=1}^{J} \lambda_j H_j$. These iid. draws can be described in terms of the indicator variable construction mentioned above as follows: by introducing "floor" indicator variables[1] $z_{1:K}$ with range $\{1, \ldots, J\}$, one for each block in the partition, which are distributed according to the discrete distribution $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_J)$, i.e. $p(z_k = j) = \lambda_j$, each table/dish ϕ_k is associated with the mixture component H_j it originated from. Given $z_k = j$, the dish ϕ_k is drawn from H_j, so that marginally $\phi_{1:K} \sim \sum_{j=1}^{J} \lambda_j H_j$ as required. The complete sequential generative process can be described as follows:

1. Each customer $i = 1, 2, 3, \ldots$ either joins an existing table or sits at a new table according to the two-parameter CRP (2.24). Let $A^{i-1} = (a_1^{i-1}, \ldots, a_{k-1}^{i-1})$

[1] The metaphor behind this is the following: Assume that the Chinese restaurant consists of multiple floors, each serving dishes from a different base distribution mixture component. Whenever a customer chooses to sit at a new table, she first chooses a floor before choosing a dish from the associated base mixture component. Which floor a table is located on is described using its floor indicator variable.

be the partition generated by the first $i-1$ customers. If a new table/block a_k is created when customer i enters,

(a) Sample the floor indicator $z_k \sim \text{Disc}(\boldsymbol{\lambda})$

(b) Sample $\phi_k \sim H_{z_k}$ from the corresponding base mixture component

2. Set x_i to the dish served at the table that customer i sat at, $x_i = \varphi(A^i, \phi_{1:|A^i|}, i)$.

This construction can be extended to the hierarchical case by noting that, as in the extension of the CRP to the CRF, draws from the base distribution can be thought of as customers entering the restaurant associated with the base distribution. In the multi-floor setting, whenever a draw is made from one of the base distribution mixture components H_{z_k} in step 1b above, a new customer enters the restaurant associated with H_{z_k}.

8.1.3 Gibbs Sampling Inference in the GPYP

The Gibbs sampler proposed by Wood and Teh [2009] for the multi-floor CRF is a slightly modified version of the remove-add sampler (Algorithm 2) for the CRF: each customer is first removed from its restaurant (and recursively from the appropriate parent restaurants if tables become empty), and then re-inserted according to the multi-floor CRP described above. The main difference to the plain CRF case arises when a customer chooses to sit at a new table: in this case, the new customer also has to sample the corresponding floor indicator variable. Wood and Teh [2009] combine these two steps into one by sampling the table for the new customer jointly with the floor indicator variable. In other words, instead of a customer of type s first deciding whether to join and existing table with probability proportional to $c_{sk} - d$ or to sit at a new one with probability proportional to $(\alpha + t_s d)\sum_j \lambda_j H_j(s)$, and then, if the second option is chosen, sampling the floor indicator from $P(z_k = j|\text{rest}) \propto \lambda_j H_j(s)$, the customer directly chooses either an existing table (with probability proportional to $c_{sk} - d$ as before), or a new table *and* the floor $z_k = j$ with probability proportional to $(\alpha + t_s d)\lambda_j H_j(s)$.

While this Gibbs sampler for the m-CRF is straight-forward and only requires us to keep track of the additional floor indicator variables, it assumes that the mixture weights λ_j are fixed. However, inferring these mixture weights from data by giving them a (hierarchical) prior makes this model much more powerful. Wood and Teh [2009] propose to do so by instead of sampling the "switch" variables z for vertex v from a fixed discrete distribution $z \sim \text{Disc}(\boldsymbol{\lambda}_v)$,

to instead sample them from a PYP *random* distribution $S_v \sim \text{PY}(\alpha_v^S, d_v^S, H_v)$, which itself is part of a hierarchical PYP prior. In this setup, each vertex v is associated with two PYP random variables: G_v, a distribution over the domain of interest (i.e. words or characters in our case), and *switch distribution* S_v, a distribution over the set $\pi(v)$ of parents of v (or equivalently over the integers $1, 2, \ldots, |\pi(v)|$). The switch distributions can themselves form a HPYP (or even GPYP) hierarchy, but in [Wood and Teh, 2009] are simply given a shared prior per context length.

Though Wood and Teh [2009] don't discuss this extension in detail, the remove-add Gibbs sampler readily also extends to this setting, by maintaining a separate Chinese restaurant franchise representation for the distributions S_v. In the basic reseating step for the G_v hierarchy described above, λ_j now becomes $S_v(j)$. Whenever a new table on floor $z = j$ is created (with probability proportional to $(\alpha + t_s d) S_v(j) H_j(s)$), a customer of type j enters the restaurant associated with S_v, where it is handled according to the same basic CRF procedure (i.e. sitting at either an already occupied or a new table, and recursively sending customers to the parent restaurant whenever new tables are created in the process). Similarly, whenever a table on floor j becomes empty in the process of updating the restaurant associated with G_v, a customer of type j is removed from the restaurant associated with S_v, recursively removing customers up the hierarchy of base distributions if necessary.

8.2 Neural Language Models

Neural language models is the name given to class of language models pioneered by the work of [Bengio et al., 2003], that have gained popularity recently due to their state-of-the art performance, often outperforming classical n-gram models by a large margin (see e.g. [Mikolov, 2012; Zaremba et al., 2014; Józefowicz et al., 2016; Dai et al., 2019; Radford et al., 2019]). The distinguishing feature of these models is that they embed the underlying discrete observations into a real-valued feature space, and that this mapping is learned from the data with the remaining parameters (unlike other feature-based language models such as maximum-entropy models [Rosenfeld, 1994], that embed the data using a hand-designed, fixed mapping). Challenges regarding the computationally expensive training of these types of models have been (partially) overcome with the development of alternative training methods [Mnih and Teh, 2012] and the widespread availability of GPUs, which are very well suited to the linear algebra computations involved.

Various types of neural network architectures that have been successfully applied in computer vision and other domains have also been applied to language modelling, including models with stochastic latent variables [Mnih and Hinton, 2007] and recurrent neural networks (RNNs) [Mikolov, 2012]. While RNNs tend to outperform purely feed-forward models, we will focus on the latter here, as these are conceptually simpler (and easier to train), and our focus here is on exploring the combination of these models with count-based models, not necessarily on achieving state-of-the-art performance.

The paper that introduced the particular class of neural language models we consider here [Bengio et al., 2003] (as well as many follow-up papers) used a fairly simple general architecture. As in n-gram models, the data consist of context-observation pairs $(\mathbf{u}, s)$, where $\mathbf{u} \in \Sigma^M$ and $s \in \Sigma$ are elements in some fixed vocabulary Σ and the goal is to model the conditional probability $P_{\mathbf{u}}(s)$ for word $s \in \Sigma$ in context $\mathbf{u} \in \Sigma^M$. Instead of modelling this dependence directly, the context and observation are mapped into *representations* in $\mathbb{R}^D$ first, through a mapping that is learned from the data (and that could be different depending on whether the word appears in the context or is the target word). Usually the context words are individually mapped first and then combined. The log-probability is then modelled as a function that measures the compatibility of the combined context representation and the target representation, e.g. the dot product. Let $\phi : \Sigma \to \mathbb{R}^D$ denote the embedding for the context words, $\psi : \Sigma \to \mathbb{R}^{D'}$ denote the embedding for the target word, $g : \mathbb{R}^{D \times M} \to \mathbb{R}^{D''}$ denote the function for combining context representations, and $h : \mathbb{R}^{D''} \times \mathbb{R}^{D'} \to \mathbb{R}$ the function for measuring compatibility, then the probability of a word s in a context $\mathbf{u} = u_1 \cdot u_2 \cdots u_M$ is given by:

$$E(\mathbf{u}, s) = h\left(g\left(\phi(u_1), \ldots, \phi(u_M)\right), \psi(s)\right) \tag{8.5}$$

$$P_{\mathbf{u}}(s) = \frac{\exp\left(E(\mathbf{u}, s)\right)}{\sum_{s' \in \Sigma} \exp\left(E(\mathbf{u}, s')\right)}. \tag{8.6}$$

In the model by Bengio et al. [2003] ϕ and ψ are simple look-up tables, g is a one-layer neural network with additional direct connections from inputs to outputs taking the form

$$g(\boldsymbol{\phi}) = \begin{bmatrix} 1 \\ \boldsymbol{\phi} \\ \tanh(d + H\boldsymbol{\phi}) \end{bmatrix}, \tag{8.7}$$

where $\boldsymbol{\phi} = [\phi(u_1), \ldots, \phi(u_M)]^T$, and $h(x, y) = \langle x, y \rangle$ is the dot product. The parameters in this model are the context representations, the target representations, as well as the weights H and biases d of the hidden layer. If the context

representations are D-dimensional and the hidden layer has h units, the total number of parameters is $|\Sigma|((m+1)D+h+1)+h(mD+1)$. The best reported test perplexities in [Bengio et al., 2003] are 276 for the Brown Corpus. Results for a model that is a mixture with an n-gram model are better: 252 for the Brown Corpus and 109 for the AP News Corpus (see Appendix B for a description of the data sets).

Learning the parameters in these models is traditionally accomplished via some variant of stochastic gradient ascent on a maximum likelihood objective function. In particular, the log-likelihood for a data set $(\mathbf{u}_i, s_i), i = 1,\ldots,N$ is given by $\ell = \sum_i \left[E(\mathbf{u}_i, s_i) - \log\sum_{s'\in\Sigma}\exp(E(\mathbf{u}_i, s'))\right]$ and the contribution of a single data point i to the gradient with respect to the model parameters θ is given by

$$\frac{\partial}{\partial\theta}E(\mathbf{u}_i, s_i) - \sum_s P_{\mathbf{u}_i}(s)\frac{\partial}{\partial\theta}E(\mathbf{u}_i, s). \tag{8.8}$$

The second term, the expected value of the gradient of the energy, is computationally problematic as its computation requires evaluation of E and its derivative for every word in the vocabulary, which can be expensive even for small vocabularies. Several approaches for overcoming this problem have been proposed, including parallelizing this computation [Bengio et al., 2003], structuring the vocabulary to avoid this expensive computation [Mnih and Hinton, 2009], and, more recently, optimizing a different objective function [Mnih and Teh, 2012] (see below).

8.2.1 Log-bilinear Model

The log-bilinear language model introduced by Mnih and Hinton [2007] is one of the simplest models in the class of feed-forward neural language models. It does not involve stochastic hidden units and simply parametrizes the log-probabilities in each context using a bilinear function in the context representations and the target word representations. The context representations are linearly transformed and then combined via addition and the compatibility of this combined context representation is then measured using the dot product. Using the notation from above, it can be specified as

$$g(\boldsymbol{\phi}) = \sum_m A_m\phi(u_m) \tag{8.9}$$

where the A_m are weight matrices for each context position $m = 1,\ldots,M$. As in the model described above, the compatibility function h is the dot product and ϕ and ψ are look-up tables (the original version in [Mnih and Hinton, 2007] used the same look-up tables for context and target words, but in [Mnih and

Teh, 2012] the authors suggested using distinct representations). Additionally, the model contains per-word biases, so that in summary

$$E(\mathbf{u}, s) = \left\langle \sum_m A_m \phi(u_m), \psi(s) \right\rangle + b_s \tag{8.10}$$

The context and target representations are typically chosen to be of the same dimensionality $D = D' = D''$, resulting in square context-weight matrices A_m.[2]

Noise-Contrastive Estimation

Training neural language models—even the simple LBL model—by maximizing log-likelihood with stochastic gradient descent can be computationally challenging. While GPUs can be used to speed up many of the computations involved in training neural language models, the reported training times are still often orders of magnitude larger than for count-based language models. As an extreme example, [Bengio et al., 2003] report a training time of 3 weeks for their model on the AP news corpus, using a parallel implementation and 40 CPUs, while a Sequence Memoizer model on the same corpus can be estimated in a under a minute on a single core.

As mentioned above, one particular bottleneck during gradient descent training using the maximum likelihood objective function is the computation of the expectation in (8.8), as it requires an amount of computation proportional to the vocabulary size [Mnih and Teh, 2012]. Mnih and Teh [2012] propose an alternative training procedure based on noise-contrastive estimation (NCE) [Gutmann and Hyvärinen, 2010], which avoids the costly computation in the gradient of the log-likelihood objective by replacing it with a different objective function. At a high level, the NCE objective function is based on the idea of training the model to optimally distinguish samples from the data from samples from a (known) noise distribution. For a given noise distribution $P^n(s)$, the resulting objective function is

$$E_{P_\mathbf{u}}\left[\frac{P_\mathbf{u}(s)}{P_\mathbf{u}(s) + kP^n(s)}\right] + kE_{P^n}\left[\frac{kP^n(s)}{P_\mathbf{u}(s) + kP^n(s)}\right] \tag{8.11}$$

where k, the number of noise samples, is a parameter of the method. We refer the reader to [Mnih and Teh, 2012] for more details. Using this optimization technique, Mnih and Teh [2012] report training times that are more than an order of magnitude faster than maximum likelihood training (from 21 hours to

[2] A further simplification that has been suggested by Mnih and Teh [2012] is to use only diagonal context weights A_m, which in combination with the noise-contrastive estimation method (described in Section 8.2.1) speeds up learning. However, we use full context weight matrices in our experiments and employ an optimized, multi-threaded BLAS implementation to speed up the costly matrix-matrix multiplications.

1.5 hours on the PTB corpus), while producing predictive performance that is on par or better than the models trained with the maximum likelihood objective. In our experiments we make use of this noise-contrastive estimation procedure for training the LBL models and model components to speed up training. We follow [Mnih and Teh, 2012] closely in our setup and use a unigram noise distribution and $k = 25$ noise samples, which was shown to strike a reasonable balance between predictive performance and computational cost in [Mnih and Teh, 2012].

Tricks for LBL Training

Training neural language models is conceptually straightforward, but some commonly used tricks are necessary in order to obtain good results. The main free parameters for LBL training are the initialization of the weights and representations, the strength of regularization, and the learning rate(s) and their decay. Discussing this in detail is beyond the scope of this thesis and we refer the reader to [Montavon et al., 2012] for a discussion of many of these issues. However, one thing we found to be particularly important when training the hybrid LBL/HPYP models is learning rate scaling (and early stopping). He were use the following simple approach also used in [Mnih and Teh, 2012]: During training, we monitor the performance (in terms of perplexity) of the current model by evaluating it on a held-out validation set after each epoch. If the perplexity increased compared to the last epoch, we scale the learning rate by a factor of $1/2$. This procedure is repeated for some fixed number of epochs, and in the end the model with the lowest validation set perplexity is chosen.

8.3 Hybrid Model

Conceptually, integrating a neural language model into a hierarchical (or graphical) PYP model is straightforward: as a neural language model can be viewed as a set of predictive distributions $\{L^{\theta}_{\mathbf{v}}(s)\}_{\mathbf{v}}$ for each context $\mathbf{v} \in \Sigma^M$ and symbol s, jointly parameterized by the network weights θ, these can be used directly as base distributions for (a subset of) the PY-distributed random distributions $G_{\mathbf{u}}$ ($\mathbf{u} \in \Sigma^K$) in a hierarchical PYP model (2.41) or the Sequence Memoizer (3.2). We will compare several possible model architectures which are summarized in Figure 8.1. Inference in these models amounts to both computing the posterior distribution over the random distributions (or the predictive distributions when the distributions are integrated out using the CRP) as well as optimizing

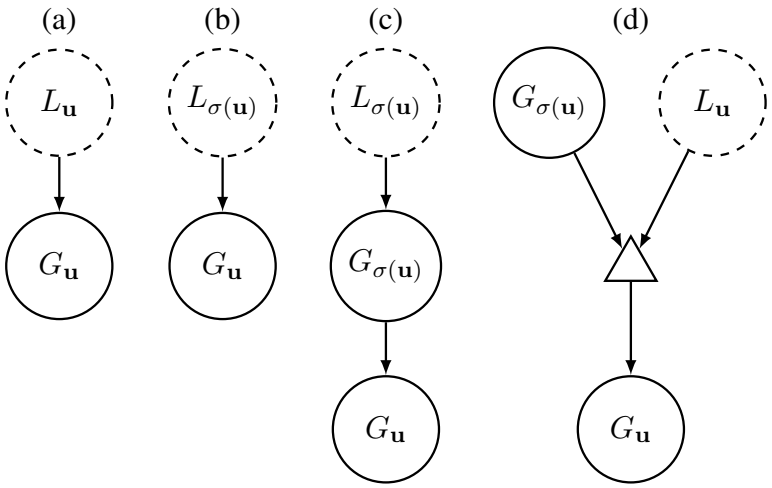


Figure 8.1: Model architectures combining a neural language model model with a (hierarchical) Pitman-Yor Process. (a) PYP used as "adaptor" for the neural model; (b) "back-off" combination; (c) combined back-off (b) with adaptor (a); (d) GPYP combination. For most experiments we considered $|\mathbf{u}| = 2$, so that (a) and(d) use an LBL model with context size 2, wheres (b) and (c) use and LBL with context size 1.

the likelihood wrt. the parameters of the neural model θ (and possibly the concentration and discount parameters).

In the most basic model of this sort we fix some context length K and let

$$G_{\mathbf{u}} \sim \text{PY}(\alpha_{\mathbf{u}}, d_{\mathbf{u}}, L^{\theta}_{\mathbf{u}}) \qquad \forall \mathbf{u} \in \Sigma^K. \tag{8.12}$$

This model does not make use of any explicit hierarchical structure, but just uses the PYP as an "adaptor" [Goldwater et al., 2006b] on top of the conditional distributions coming from the neural model. As we are using the LBL as the neural language model, we call this model LBL/A (log-bilinear model with adaptor). Considering the CRPs associated with each of the $G_{\mathbf{u}}$, we can see that the "training data" for the neural language model are the dishes served at the tables (i.e. the draws from the base distribution) in each CRP. Intuitively, as the number of tables will be smaller than the number of customers (i.e. the original observations) both in expectation and at the most likely configuration, this tends to "flatten" the conditional distributions of words that the LBL is trained on towards a uniform distribution over the words observed in each context $\mathbf{u}$. Indeed, if inference were performed using the Kneser-Ney approximation to inference in PYP models, the training data for the LBL model would simply be context/observation pairs $(\mathbf{u}, s)$ for each $c_{\mathbf{u}s} > 0$. In other words, no matter how many times the symbol s is observed in context $\mathbf{u}$, the LBL will be trained as if it only occurred once.

A variation on this model, emphasizing the hierarchical aspect over the adaptor aspect, is given by

$$G_{\mathbf{u}} \sim \mathrm{PY}(\alpha_{\mathbf{u}}, d_{\mathbf{u}}, L_{\sigma(\mathbf{u})}) \qquad \forall \mathbf{u} \in \Sigma^K \tag{8.13}$$

so that subsets of the $G_{\mathbf{u}}$ that share the same suffix $\sigma(\mathbf{u})$ share the same LBL-based base distribution. The idea underlying this model is that the LBL tends to perform well when short context lengths (1 or 2) are used, but does not benefit much from using longer contexts [Mnih and Hinton, 2007], whereas count-based models see significant improvements in predictive performance for longer contexts. We will not further consider this architecture in this form, but instead focus on the following combination of these two ideas, that keeps both the hierarchical aspect as well as using a PYP adaptor for the LBL:

$$G_{\mathbf{u}} | G_{\sigma(\mathbf{u})} \sim \mathrm{PY}(\alpha_{\mathbf{u}}, d_{\mathbf{u}}, G_{\sigma(\mathbf{u})}) \qquad |\mathbf{u}| > M \tag{8.14}$$

$$G_{\mathbf{u}} \sim \mathrm{PY}(\alpha_{\mathbf{u}}, d_{\mathbf{u}}, L_{\mathbf{u}}) \qquad |\mathbf{u}| = M. \tag{8.15}$$

Note that in this model each LBL conditional distribution is directly used as base distribution for a single $G_{\mathbf{u}}$, which in turn is used as base distribution for $G_{s\mathbf{u}}$ for all $s \in \Sigma$, so that in comparison to the model described above there is an additional level of PYP adaptors. Further, this kind of model can be extended to deeper hierarchies with $K > M+1$, where for context lengths larger than M the model has the standard HPYP structure.

Finally, we can integrate the LBL model using the *graphical Pitman-Yor process*, by making the base distribution $H_{\mathbf{u}}$ for a Pitman-Yor distributed random measure $G_{\mathbf{u}}$ a mixture distribution between the HPYP parent node $G_{\sigma(\mathbf{u})}$ within the context tree and the LBL distribution $L_{\mathbf{u}}$ at some context length M and using the usual HPYP hierarchy otherwise:

$$G_{\mathbf{u}} \sim \mathrm{PY}(\alpha_{\mathbf{u}}, d_{\mathbf{u}}, H_{\mathbf{u}}) \tag{8.16}$$

$$H_{\mathbf{u}} = \begin{cases} \lambda_{\mathbf{u}} G_{\sigma(\mathbf{u})} + (1-\lambda_{\mathbf{u}}) L_{\mathbf{u}} & \text{if } |\mathbf{u}| = M \\ G_{\sigma(\mathbf{u})} & \text{if } \mathbf{u} < M. \end{cases} \tag{8.17}$$

One can easily extend this setup further to other architectures, e.g. involving multiple LBL models at different context lengths, adding an additional layer of adaptor nodes, or adding other further types of language models, but we restrict our experiments to the model types described above.

As in the original application of the GPYP to domain adaptation as well as the simple mixture model (8.1), the value of the mixing weight $\lambda_{\mathbf{u}}$ has a significant influence on the performance of this model. We follow Wood and Teh [2009] and use a hierarchical prior on the mixing weights as well (as described

above) and perform inference over the hierarchy of indicator variables using Gibbs sampling.

8.4 Inference & Learning

Given a training corpus, inference and learning in the proposed models amounts to both fitting the parameters of the LBL model as well as inferring the posterior distributions (or the most likely posterior state) over the latent variables in the hierarchical (or graphical) Pitman-Yor process.

If the parameters θ of the LBL model were fixed, the approaches for inference in (hierarchical) PYP models discussed in Chapters 2 & 3 could directly be applied to the non-GPYP models, and the multi-floor CRF Gibbs sampler of Wood and Teh [2009] could be used to infer the seating arrangements and floor indicator variables of the GPYP model. Conversely, if we were given a seating arrangement in the CRF and the settings of the m-CRF indicator variables (for the GPYP), we knew which dishes were assigned to the LBL component and could optimize the LBL model component based on these by either using a gradient-based maximum-likelihood method or noise-contrastive estimation.

Based on this observation, our inference and learning approach alternates between sampling the seating arrangements and indicator variables of the HPYP/GPYP component using Gibbs sampling, and optimizing the parameters of the LBL model using NCE. In all reported experiments we interleave one iteration of Gibbs sampling (i.e. reseating all customers in all restaurants) with one iteration of NCE updates for all counts assigned to the LBL component.[3] The proposed approach can be seen as form of an incremental EM algorithm, where the expectation step is approximated by a "single sample" Monte Carlo approximation to the true posterior distribution, and the (partial) maximization step is performed using a series of NCE updates. The use of a single sample to approximate the expected value required for an optimization is akin to the Contrastive Divergence [Hinton, 2002] algorithm which has been successfully used for training e.g. restricted Boltzmann machines.

In a similar fashion, we can additionally interleave gradient-based updates to the hyperparameters of the HPYP/GPYP model component, i.e. the concentration and discount parameters, as mentioned in Section 5.7.2 for the SM model. While due to the approximations made this algorithm enjoys no theoretical guarantees of convergence or optimality, in practice it appears to

[3] We did not perform extensive experiments comparing different schedules, as ad-hoc experiments did not hint at consistent improvements by either taking more steps in the sampler or performing more optimization steps for the LBL component.

reliably converge to states of similar predictive performance when started from different initializations.

We also explore a simpler version of the training procedure (for the HPYP-based models) where the table counts $t_{\mathbf{u}s}$ in HPYP component are deterministically set using the Kneser-Ney approximation (cf. Seq 2.4.2), i.e. no additional sampling needs to be performed.

8.5 Results & Analysis

In order to quantitatively and qualitatively explore the performance of the hybrid models, we performed several experiments on the Penn Treebank corpus.

We focus on models with shallow trees (i.e. contexts lengths one and two), as this is where the differences between the models can be most easily observed. When using models with longer contexts for the LBL part (e.g. a HPYP tree of depth 4 with a LBL base measure at depth 4), many context/symbol pairs will only be observed once, so that these models become similar to a plain LBL model without the PYP component. Conversely, increasing the depth of the HPYP component while using small context lengths for the LBL will mainly demonstrate the ability of the HPYP to make use of longer contexts, which was already explored in Chapter 4.

8.5.1 Predictive Performance

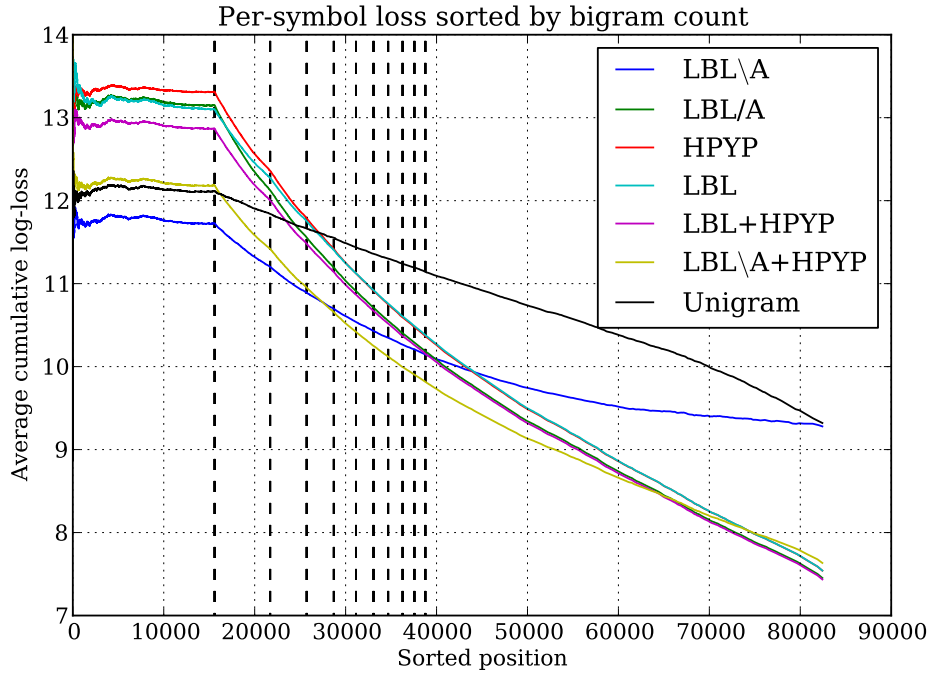
The first set of experiments asses the predictive performance of the presented models in terms of perplexity on the Penn Treebank test set. All models were trained using the iterative training procedure discussed above for 100 iterations or until the learning rate for the LBL model decreased below $1e{-}10$ (the learning rate was initialized at 0.1 and halved every time the validation set performance stopped improving). The LBL model weights were initialized randomly and the dimensionality of the LBL representations was fixed at $D = 100$ for these experiments.

Table 8.1 summarizes the results. Several observations can be made: All of the hybrid model variants outperform the individual component models (using the same context size). While taken individually an LBL model with context size $M = 2$ significantly outperforms a model with context size $M = 1$ (perplexity 160 vs. 186), the combined models tend to perform better when the $L = 1$ LBL is combined with a depth $K = 2$ HPYP/GPYP model. The GPYP models perform worse than their HPYP counterparts (though still outperforming the individual baselines), though this can be partially remedied by initializing the LBL com-

ponent in the hybrid GPYP models with a pre-trained LBL state (see below). Performing inference in the HPYP component (instead of setting the counts deterministically using the Kneser-Ney approximation) provides a significant performance improvement. For the trigram $K = 2$ models, a linear interpolation (8.1) of the two pure base models trained separately outperforms the hybrid models (perplexity 129.1 with optimal weights chosen on the validation set). For the bigram models, the test set perplexity of the linear interpolation is very similar to that of the PYP adaptor model.

It is revealing to study the individual models' strengths and weaknesses in order to understand why combinations of them—in form of mixtures or the HPYP/GPYP models presented here—tend to outperform the individual components by a large margin. Figure 8.2 shines some light on this question (focussing on the bigram HPYP-based models): it shows the average cumulative log-loss, i.e. $\frac{1}{i}\sum_{j=0}^{i} -\log \mathrm{P}(s_j \mid \mathbf{u}_j)$ as a function of i, where $(\mathbf{u}_j, s_j)$ are context-symbol pairs from the test set, sorted by the number of times they occur in the training set in ascending order. This type of plot allows us to examine how the predictive performance on the test set breaks down into losses incurred at individual positions, as a function of how often a particular symbol-context pair was observed during training. The head of the curves on the left reflects the performance for contexts where little training data is available, while the tail shows how well the model performs on contexts with more occurrences in the training data. A few interesting observations can be made here: The interval to the left of the first dashed line contains context-symbol pairs that did not occur in the training data at all. In order to make predictions for these, the count-based models fall-back to their unigram models, so that the non-smoothed unigram model shown in black represents a lower bound for these models in this regime. The plain bigram HPYP model shown in red performs poorly in this regime, while the LBL model trained with a PYP adaptor but where the adaptor is not used during prediction (`PYP\A`), shown in blue performs best and outperforms both the unigram baseline and the LBL model trained on the raw data, demonstrating (a) that using the "adapted", flattened counts for training the LBL model improves its performance in this regime; and (b) that the LBL model can effectively leverage the learned word representations to generalize to unseen context-symbol pairs. The combined adaptor model, as well as mixtures between the plain LBL and the adapted LBL lie between these extremes. For context-symbol pairs that occur more frequently in the training data the picture is mostly reversed: the unigram baseline unsurprisingly performs poorly, as does the LBL component trained with adaptor when

(a)



(b)

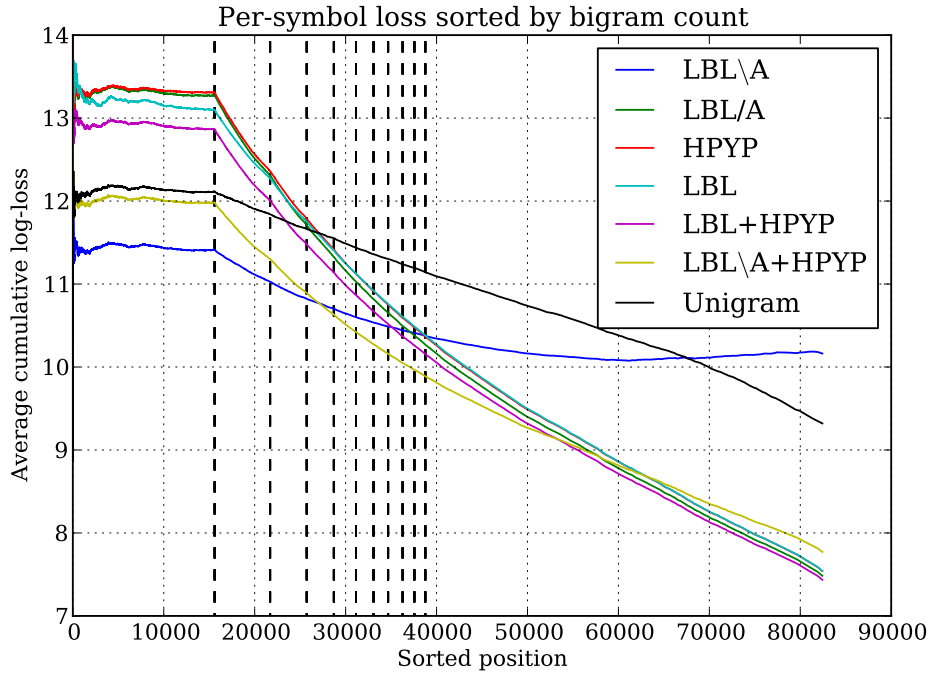


Figure 8.2: Average cumulative log-loss, i.e. $\frac{1}{i}\sum_{j=0}^{i} -\log p(s_j \mid \mathbf{u}_j)$ as a function of i, where $(\mathbf{u}_j, s_j)$ are context-symbol pairs from the test set, sorted by the number of times they occur in the training set in ascending order. Vertical dashed lines mark $0, 1, 2, \ldots, 10$ occurrences in the training set. The top panel (a) shows models where inference in the HPYP component is performed using Gibbs sampling. Panel (b) shows models where the HPYP component is deterministically set using the Kneser-Ney approximation. The different methods are named as follows: (1) `LBL\A` – bigram LBL model trained through a PYP adaptor, but predictions made directly from the LBL; (2) `LBL/A` – as (1), but predictions made through PYP adaptor; (3) `HPYP` – plain bigram HPYP model; (4) `LBL` – plain bigram LBL model; (5) `LBL + HPYP` – Linear interpolation of (3) and (4); (6) `LBL\A + HPYP` – Linear interpolation between (1) and (3); (7) `Unigram` – MLE Unigram model

HPYP Models

Tree Depth K	LBL depth M	KN approx.	Perplexity
1	1	yes	178.9
1	1	no	175.2
2	1	yes	146.4
2	1	no	141.8
2	2	yes	145.8
2	2	no	143.5

GPYP Models

Tree Depth K	LBL depth M	Perplexity
2	1	144.9
2	2	149.7

Baseline Models

Model	Context Size	Perplexity
LBL, $D = 100$	1	186
LBL, $D = 100$	2	160
HPYP-1PF	1	186
HPYP-KN	1	186
HPYP-1PF	2	150
HPYP-KN	2	151

Table 8.1: Perplexity results of various LBL/HPYP hybrid models on the Penn Treebank corpus test set. For models where KN=yes, the Kneser-Ney approximation (Sec. 2.4.2) was used to infer the counts for the HPYP part, i.e. only the LBL component was optimized; the KN=no models were trained using the alternating EM-like algorithm using Gibbs sampling for the HPYP counts. For all these model the LBL representation dimensionality is $D = 100$. The top table contains results for the HPYP models, with models where $K = M$ refer to model type (a) and models where $K = M+1$ refer to model type (c). The bottom table contains results for GPYP models (type (d)). For the GPYP models the full alternating inference procedure over counts, indicator assignments, mixing weights, and LBL parameters was used, with randomly initialized weights.

the adaptor is not used during prediction. Interestingly, the plain HPYP and LBL models have virtually identically performance for context-symbol pairs that occur more than three times, which implies that the HPYP makes more effective use of observations that occur one to three times (the red and cyan lines converge between the first and third vertical line). Finally, in this setting, an equally-weighted mixture between separately trained LBL and HPYP models has a final performance that is indistinguishable from the jointly-trained

model.

The top panel of Figure 8.2 shows the performance of models where the counts of the HPYP component are inferred using Gibbs sampling (only the last sample is used, i.e. no averaging across samples), and the bottom panel shows results were the counts were deterministically computed using the Kneser-Ney approximation, and only the LBL parameters and the hyperparameters were optimized. Interestingly, the LBL\A model performs better on unseen observations when the KN approximation is used, but the final performance of such a model is significantly worse. Further, the final perplexity of the KN-based LBL/A model is worse than that of the linear interpolation LBL+HPYP, while in the Gibbs sampled setting it is on par.

8.5.2 Initialization & Pre-Training

As our alternating optimization and inference method is not guaranteed to converge to the globally optimal parameters, we empirically evaluate the effect of initialization on the predictive performance. For the HPYP-based models, pre-training the LBL component either on raw data or on modified counts obtained from the Kneser-Ney approximation did not significantly improve (or degrade) performance, with models achieving similar performance after convergence. However, for the GPYP-based models, initializing the LBL component randomly seems to have a detrimental effect: if the LBL component is initialized randomly the predictive distributions are initially effectively uniform, so that the initial stages of inference of the indicator variables will assign most of the observations to the HPYP component, which in turn will prevent the optimization of the LBL component from improving its accuracy. For a depth-2 GPYP/LBL model, randomly initializing the weights for the LBL component leads to a final test set perplexity of 149.7. Pre-training the LBL component on the raw data until convergence (achieving a test set perplexity of 160.0) before starting the Gibbs sampling inference procedure for the GPYP component leads to an improved final perplexity of 145.5 (for the last sample, 142.9 if the last 10 samples are averaged).

8.5.3 Varying the Representation Dimensionality

For the hybrid LBL/PYP models, the LBL component is trained on significantly less data than an LBL model trained by itself. For example, for the WSJ corpus, of the 920k training tokens only about 370k are used to train an LBL at context length 1. However, the described LBL model with $D = 100$ has about two

million parameters, and thus can significantly overfit the data. The learning rate adaptation and early stopping procedures described in Section 8.2.1 mitigate this problem somewhat, but shrinking the complexity of the model with the size of the available training seems desirable. Figure 8.2 shows the performance of the best-performing model from Table 8.1, i.e. the HPYP combination with $M = 1$ and $K = 2$, as the representation dimensionality is varied.

	10	20	30	40	50	60	70	80	90	100
Perplexity	140.2	140.9	141.5	142.0	142.4	142.7	143.1	143.1	141.2	141.8

Table 8.2: Perplexities on the WSJ corpus of an LBL/PYP model with total context length $K = 2$ and LBL context length $M = 1$ as the representation dimensionality D is varied.

8.5.4 Conditional Distribution

Finally, we qualitatively investigated how the conditional distributions learned by the different models compare. Figure 8.3 shows the conditional distribution following the word "the", which is the most frequent token in the corpus, and Figure 8.4 shows the conditional distribution following "now", which is the 100-th most frequent token. In both plots, the black line shows the conditional distribution of a non-smoothed bigram model. The area to the right of the plot, where the black line is not visible, corresponds to the symbols that did not follow "the" or "now" in the training set, respectively. The other models all assign non-zero probability mass to these zero-frequency types, which they necessarily need to take away from some of the more frequently occurring tokens. For the more frequently occurring context token "the", the conditional distributions are almost identical to the unsmoothed bigram distribution, with the exception of–as expected–the `LBL-1` model which has been trained skewed towards a uniform distribution by the PYP adaptor. For the less frequent "now" conditional distribution, the re-distribution off probability mass to the zero-frequency types is more pronounced, as there are more of these types. Interestingly, the LBL and HPYP models re-distribute the probability mass somewhat differently: The LBL assigns higher mass to the high-frequency tokens, and deducts this mass from the types the occur with frequency one and two, while for the HPYP and combined HPYP/LBL model this effect is less strong. Finally, it can be seen the the conditional probabilities assigned by the LBL models are more variable in the low training frequency buckets, indicating that the models are less reliant on the conditional counts in this regime.

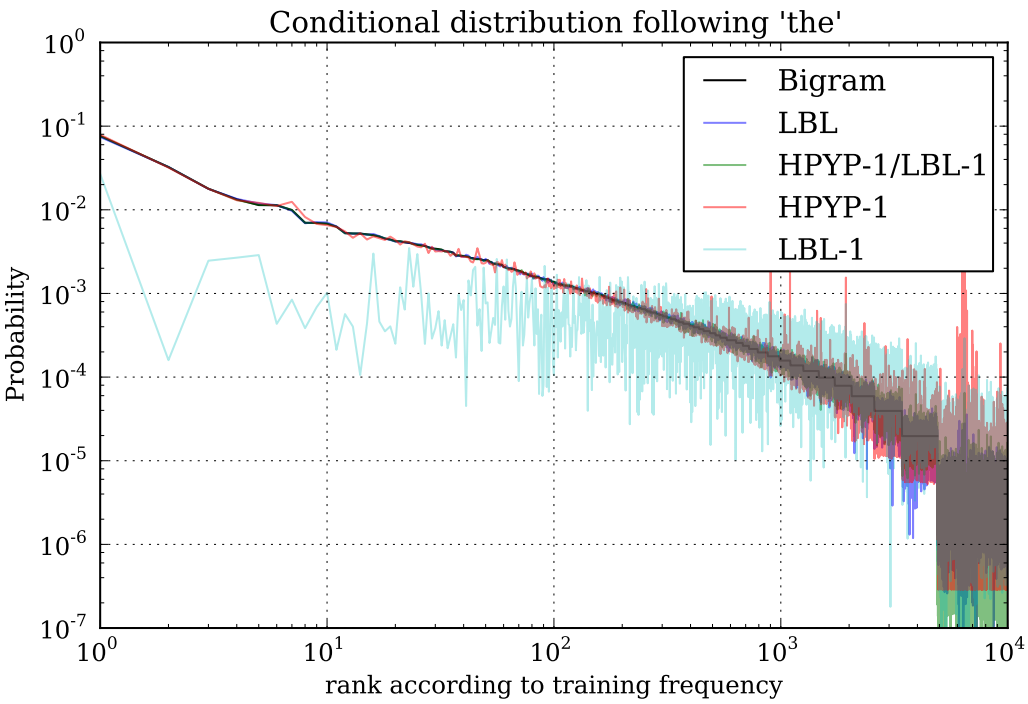


Figure 8.3: Learned predictive distribution following the context `the`, which is the most frequent token in the corpus. Models were trained on the WSJ corpus. The words are ordered on the x-axis according to their (relative) frequency following `the` in the training sequence, which is shown in black. LBL denotes a plain LBL model, HPYP-1/LBL-1 a hybrid model with tree depth and LBL context length both 1, HPYP-1 a plain HPYP model with context length 1, and LBL-1 is just the LBL component of the HPYP-1/LBL-1 model.

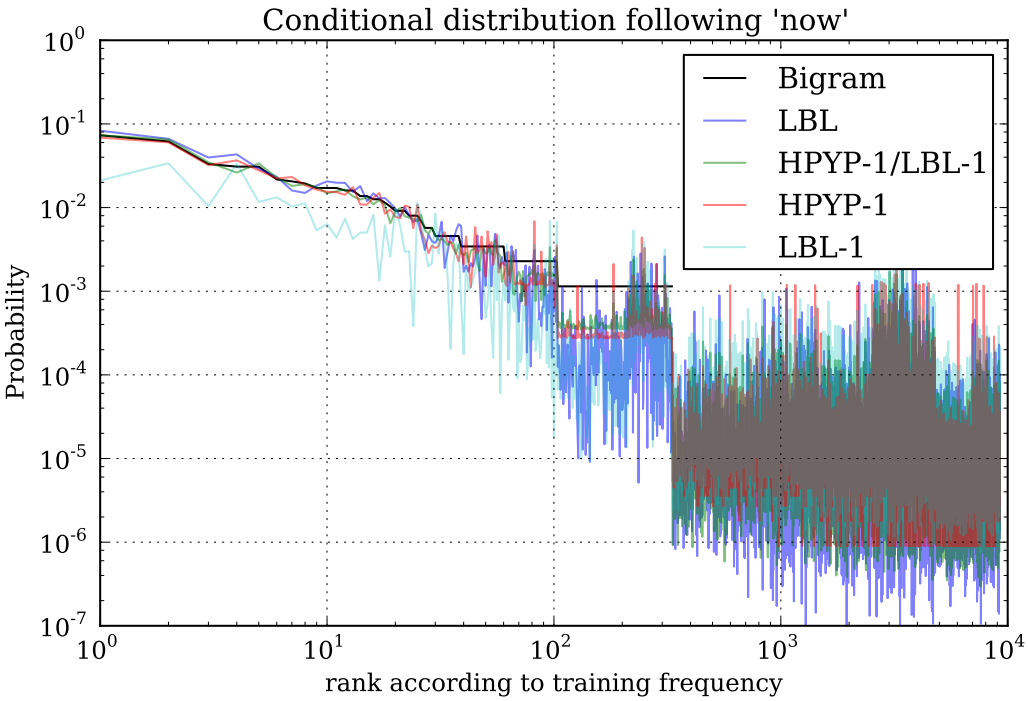


Figure 8.4: Learned predictive distribution following the context `now`, which is the 100-th most frequent token in the corpus. The models are the same as in Figure 8.3

8.6 Discussion

We have explored several hybrid model variants combining a LBL neural language models with variants of hierarchical PYP models. While the predictive performance of the specific models considered is underwhelming (i.e. they do not outperform simple linear interpolations in terms of test set perplexity), the general direction of combining count-based on neural language models is still an interesting avenue for further exploration.

Most of the work on the model described in this chapter was performed around 2012, when (non-recurrent) neural language models such as the LBL model incorporated here were showing very promising results, but the improvements neural language models would bring to the field over the next couple of years was not clear yet. Neural language models were still a niche and suffered from drawbacks such as slow training, the requirement to use small vocabularies, and the necessity to blend the model with an n-gram model to achieve state-of-the-art performance.

Since then, the language modelling landscape has shifted quite dramatically, and modern recurrent neural language models outperform the models presented here by a large margin. For example, Zaremba et al. [2014] report a perplexity of 68.7 on the WSJ/Penn Treebank corpus for an ensemble of LSTM-based RNN models. This success of recurrent neural networks for language modelling, started by the work of Mikolov [2012] has not only lead to the integration of such language models into downstream NLP tasks and systems, but also to the development of end-to-end models for such tasks based on recurrent neural networks (e.g. neural machine translation [Bahdanau et al., 2014; Popescu-Belis, 2019] and speech recognition [Graves et al., 2013]). The idea of using deep learning models to solve "NLP from scratch" [Collobert et al., 2011], i.e. replacing complex pipelines with models based on word representations obtained through unsupervised language modelling, has also been very successful, exemplified e.g. by the BERT [Devlin et al., 2018] and ELMo [Peters et al., 2018] embeddings.

In spite of the great success of such "purely neural" models, we do believe there is value in studying combinations of count-based models, which explicitly encode and exploit structure, and neural models, which induce representations of syntactic and semantic patterns, as both have strengths and weaknesses. While neural models appear to be able to model and reproduce the power-law behavior present in natural language [Takahashi and Tanaka-Ishii, 2017], explicitly encoding this power-law behavior in the model might allow it to devote more of its capacity to modelling the remaining structure.

The benefits of such hybrid approaches when combined with modern deep learning techniques have recently been demonstrated by Neubig and Dyer [2016], who have shown that a different way of integrating count-based and neural language models can be very effective: by either parametrizing the mixing weights in a mixture language model using a neural network, or by combing this technique with a feed-forward neural language model they are able to achieve state-of-the art results.

8.7 ScaNPB – Software Framework for PYP-based Nonparametric Bayesian Models

The model architectures presented in this and the preceding Chapters are just a small subset of the models that can be constructed from hierarchical or graphical Pitman-Yor processes as fundamental building blocks. Implementing the model construction and inference procedures (e.g. Gibbs sampling or particle filtering) for each new model architecture can be time-consuming and error prone, making it costly to explore and experiment with new models. In order to alleviate this burden (and to perform the described experiments), we implemented a generic toolkit for constructing and performing inference in PYP-based nonparametric Bayesian models. This software, the *Scala Nonparametric Bayes* toolkit, `ScaNPB` if freely available under an open-source licence.[4] Mature software packages for probabilistic modelling exist for other classes of probability distributions, e.g. Stan [Carpenter et al., 2017] (for parametric models), or Edward [Tran et al., 2017] (for deep learning-based models and inference techniques), but they either do not support nonparametric Bayesian models, or are not scalable to models with hundreds of thousands of random variables (or both). For example, while generic probabilistic programming languages such as Church [Goodman et al., 2012] or Anglican [Wood et al., 2014] support nonparametric Bayesian models, they rely on generic inference algorithms that do not scale to the models considered here.

`ScaNPB` is written in the Scala programming language, which provides high performance (being a compiled language running on the JVM) while at the same time providing the programmer with high-level abstractions yielding high development speed.

In `ScaNPB`, models are constructed by Scala programs that describe that random variables in the model and the relationships between them. This description of the model structure is kept separate from the inference algorithm

[4] https://github.com/jgasthaus/scanpb

and any *state* that may need to be kept for each random variable during inference (e.g. some CRP representation). This separation makes both models as well as inference procedures *composable*, allowing them to be built up from smaller building blocks. Each *node* in the model (representing a random variable) can be assigned an inference (or optimization) procedure. Making models and inference procedures composable has been identified as a desireable feature in other work on software for probabilistic modelling, such as Edward [Tran et al., 2017] and Pyro[5].

ScaNPB contains generic code for handling context tries and trees, making it easy to construct the hierarchical models described in this thesis. To illustrate how models are constructed in ScaNPB, let us consider a fixed-depth HPYP model. The model is constructed by first constructing the context trie out of nodes objects that also mix in the PYPRandomVariable trait, and then setting the base measure of each node to its parent in the trie, as well as setting the discount and concentration parameters (to shared parameters in this example). Finally, we assign MCMC sampling procedures to parts of the model by adding a mapping from node indices to samplers to the model:

```
class ContextTreePYPNode[A](val slice: Slice)
    extends ContextTreeProbabilityMeasureNode[A]
       with PYPRandomVariable[A]

def buildModel(seq: Seq[Int], depth: Int, numTypes): Model = {
  // construct the context trie
  val root = SuffixTrie.buildForSeq(seq, depth,
                                    ContextTreePYPNode[Int](_))

  val rootMeasure = UniformIntegerProbabilityMeasure(numTypes)
  val concentration = PositiveRealParameter(1.0)
  val discount = UnitRealParameter(0.5)

  // hook up dependencies between variables
  Tree.preorderIterator(root).foreach { n =>
    n.baseMeasure = n.parent.getOrElse(rootMeasure)
    n.concentration = concentration
    n.discount = discount
  }

  val treeNodes = Tree.preorderIterator(root).toIndexedSeq
  val otherNodes = Seq(rootMeasure, concentration, discount)
  val model = Model(treeNodes ++ otherNodes)
```

[5] https://pyro.ai

```
  // assign MCMC sampling inference
  val mhSampler = MHSampler(defaultRNG,
    new GaussianProposal(Seq(0.1, 0.05)), false)
  val pypSampler = new ReseatingSampler[Int](defaultRNG)

  model.samplers ++= Seq(
    pypSampler -> treeNodes.map(model.nodeIndex.apply(_))
    mhSampler -> otherNodes.map(model.nodeIndex.apply(_)))

  model.assignIndices() // finalize model construction
  model
}
```

In addition to models and their building blocks (`Nodes` and `Variables`), the second core concept in `ScaNPB` is that of *state*, which is maintained separately from the model. There are two motivations for this separation between model and state: First, different implementations of the state interface for a given random variable type can be used by the same generic inference code. For example, the generic remove/add sampler for PYP random variables (Algorithm 2) can be used with state classes implementing different CRP representations (Chapter 4). Second, the state can be mutated and saved independently from the model, making it easy to implement operations such as checkpointing, particle filtering (maintaining multiple different copies of the state), or global Metropolis-Hastings updates, which require an old state to be reconstructed when a proposal is rejected.

Samplers in `ScaNPB` typically define a default state implementation for the node types they can operate on, so that creating a default state object is easy:

```
val buffer = new Array[State](model.nodes.length)
 model.samplers.foreach { case (sampler, idxs) =>
  idxs.foreach { idx =>
    buffer(idx) = sampler.getInitialState(model, idx)
  }
state = ImmutableMultivariateState(s)
```

Samplers implement a `step()` method that transforms a state into and updated state. When multiple samplers are defined on a given model, we can easily run them sequentially by using a `fold()`:

```
def sample(model: Model, state: MultivariateState): MultivariateState = {
  model.samplers.foldLeft(state) { case (s, (sampler, indices)) =>
    sampler.step(model, s, indices)
  }
}
```

To complete the HPYP example, we need to further ingredients. Before running the Gibbs sampling-based inference procedure, we need to add the customers corresponding to our observations into the state. This is achieved by finding the context node in the tree and using the `AddCustomerKernel` (which is a building block of the remove/add sampler) to modify the state:

```
val kernel = AddCustomerKernel(defaultRNG)
seq.indices.foldLeft(state) { case (s, i) =>
  val (node, slice) = SuffixTree.findInsertionPoint(
      seq,
      root,
      Slice(0, i)
  )

  kernel.step(model, s, node.idx, seq(i))
}
```

Finally, after obtaining one or multiple states representing the posterior distribution of interest, we can use the state to make predictions. In the HPYP model example, for a given new context-symbol pair, this amounts to finding the corresponding node in the the context tree and then using the local state associated with that node to make a prediction:

```
def predictSingle(seq: Seq[Int],
                  ngram: Seq[Int],
                  root: ContextTreeProbabilityMeasureNode[Int],
                  model: Model,
                  state: MultivariateState): Double = {
  val node = SuffixTrie.findLongestSuffixSeq(
      seq,
      root,
      ngram.slice(0, ngram.length - 1)
  )
  val localState = state(node.idx).asInstanceOf[MeasureState[Int]]
  localState.measure(model, state, node.idx, ngram.last)
}
```

Further examples, as well as code implementing the models discussed in this thesis can be found in the `scripts` directory in the source code.

CHAPTER 9

Summary & Conclusions

9.1 Summary

In this thesis we have considered probabilistic models for sequences of symbols that explicitly incorporate prior knowledge about structural and distributional properties of the sequences. In particular, we have constructed various models from hierarchies of random probability measures following a Pitman-Yor process, thus explicitly encoding assumptions about the power-law nature of the marginal and conditional distributions, as well as encoding hierarchical relationships between distributions following different context sequences.

The bulk of the work presented here focussed on how to make inference and learning in such models *practical*, by exploring various approximate inference techniques and associated space-efficient model representations and construction algorithms, some of which can be used in an online setting, where model construction and inference happen sequentially as new data is observed. The resulting techniques are fast enough to build models for sequences with millions of symbols in a matter of seconds on commodity hardware.

In addition to the more traditional language modelling setting, we have considered an application of such models and associated online approximate inference techniques to the task of lossless data compression by coupling a Sequence Memoizer model to an entropy coder. The very promising results in this direction have led to some interesting follow-up work [Bartlett et al., 2010; Bartlett and Wood, 2011; Steinruecken, 2014], as well as the founding of a startup company[1] to commercialize this technology.

In the final two chapters we have explored ways of going beyond hierarchical PYP models by incorporating other model structures and distribution

[1] `http://deplump.org` founded by Frank Wood, who was a co-author on some of this work.

types. Chapter 7 explored how PYP-based models that incorporate a form of caching for modelling burstiness and non-stationarity can be built from the same building blocks developed in the first half of the thesis. The hybrid HPYP/LBL models described in Chapter 8 are one attempt to integrate the strengths of neural language models and PYP-based hierarchical models. While more work in this direction is needed to fully explore the potential of such models, especially when combined with recent, more complex neural language model architectures (e.g. based on recurrent, convolutional, or transformer networks), we believe that the initial results demonstrate that the models have complementary strengths and that combining them in the PYP framework (e.g. by using the proposed alternating optimization scheme) is one feasible approach for such an integration.

We have implemented the methods described in this thesis in two open source software packages, `libPlump` and `ScaNPB`, that can be used as basis for applications of these techniques or further research using these or similar classes of models.

9.2 Future Work

There are several concrete directions along which the work presented here could be continued and extended. On the theoretical side there are interesting open questions about the consistency and universality properties of Sequence Memoizer-style models with unbounded context length. For example, for the Context Tree Weighting algorithm [Willems et al., 1995; Willems, 1998] it has been shown that the method is universal for FSMX sources, in the sense that an entropy coding-based compression scheme based on this method converges to the optimal rate if the true source is an FSMX source. It would be interesting to explore whether a similar result can be established for models similar to the Sequence Memoizer.

On the algorithmic side, one open question is whether it is possible to obtain a better approximation to the most probable setting of the "multiplicity" variables $t_{\mathbf{u}s}$ under the posterior distribution of an HPYP model. While the particle filter or fractional counts-based approximations can be used to obtain settings of these variables that lead to good predictive performance without requiring an iterative procedure, it would be interesting to explore whether a scheme that more directly approximates the MAP solution (as we attempted in Section 5.5 without success) can be constructed. Another somewhat unfortunate property of all described inference techniques is that they become slower

and less accurate as the discount parameter approaches one, i.e. as the distribution becomes more similar to the base distribution. Intuitively one would prefer the opposite behavior, where modelling becomes easier the closer the distribution is to the base distribution. It is thus interesting to explore whether inference techniques and/or representations for the HPYP model class exist that have this property, or whether a model class that with similar attributes that has this property can be found.

More broadly, one could explore other classes of non-parametric (or parametric) distributions that allow modelling the power-law (or other) properties of the data more closely, or are more amenable to fast (yet accurate) approximate inference.

The Sequence Memoizer language model has state-of-the-art performance among count-based language models. However, recently the performance gap between these models and language models based on deep learning techniques has widened considerably. While a Sequence Memoizer model without extensive hyperparameter tuning (e.g. using the default configuration in `libPlump`) achieves a perplexity of around 140 on the WSJ data set and thus outperforms a 5-gram model with Kneser-Ney smoothing (perplexity 141.2 reported in [Mikolov, 2012]), the performance of RNN-based models has rapidly improved (from 113.7 reported in [Mikolov and Zweig, 2012] to 78.4 in [Zaremba et al., 2014], to 52.8 (using a form of neural cache model) in [Merity et al., 2017]. In light of this, one question of practical importance is whether any of the techniques developed here can be brought to bear on this class of models to improve this impressive performance even further. Whether such a fusion of techniques can be successful by explicitly combining both model classes (as we explored with simple models in Chapter 8), or whether a different approach for incorporating structural and distributional assumptions into neural language models is necessary to improve performance even further is entirely open and an exciting area for research.

More generally, while deep learning-based techniques have been very successful at addressing many highly relevant machine learning problems in domains like computer vision, natural language processing, and reinforcement learning, practical methods for solving real-world problems often require prior knowledge, for example in the form of structural or distributional assumptions, to be encoded in the model to be data-efficient or to guarantee predictable and reliable behavior in regimes where little training data is available. How to best combine the strengths of deep learning-based techniques with models that encode explicit prior assumptions, or how to encode such assumptions

in a principled way in neural network models directly, is an interesting broad direction for future research.

Appendices

APPENDIX A

Additional Background Material

A.1 Kramp's Symbol / Rising Factorial

The rising factorial (also called the Pochhammer symbol) and its generalization Kramp's symbol appear frequently in equations for the probability of partitions under the CRP. Kramp's symbol is defined as

$$[b]_\alpha^n = \prod_{i=0}^{n-1}(b+i\alpha) = b(b+\alpha)(b+2\alpha)\cdots(b+(n-1)\alpha) \tag{A.1}$$

i.e. it is a rising factorial with increment α. The Pochhammer symbol $(b)_n$ amounts to the special case where $\alpha = 1$, and they are related by

$$[b]_\alpha^n = \alpha^n [b/\alpha]_1^n = \alpha^n (b/\alpha)_n. \tag{A.2}$$

This relation to the Pochhammer symbol is useful because its direct definition in terms of the Gamma function $(b)_n = \Gamma(b+n)/\Gamma(b)$ yields a formula that is useful for computation:

$$[b]_\alpha^n = \alpha^n \frac{\Gamma(n+b/\alpha)}{\Gamma(b/\alpha)}. \tag{A.3}$$

In implementations, the division should be implemented as a subtraction in the log domain if needed to avoid numeric overflow (see e.g. the implementation of `gsl_sf_poch` in the GNU Scientific Library [Gough, 2009].) An alternative notation that is sometimes used (e.g. in [Pitman, 2002]) is $(b)_{n\uparrow\alpha} = [b]_\alpha^n$.

A.2 Generalized Stirling Numbers

The generalized Stirling numbers $S_d(c,t)$ that appear in many expressions related to the two-parameter CRP in this thesis are generalized Stirling numbers of type (-1, -d, 0) according to the classification of Hsu and Shiue [1998]. They

appear as the normalization constant in the CRP distribution over partitions with a fixed number of blocks and can be given an explicit characterization as the sum

$$S_d(c,t) = \sum_{A \in \mathscr{A}_{ct}} \prod_{a \in A} [1-d]_1^{|a|-1}. \tag{A.4}$$

It can also be given the following recursive definition [Hsu and Shiue, 1998; Teh, 2006a]

$$S_d(c,t) = S_d(c-1,t-1) + (c-1-dt)S_d(c-1,t) \tag{A.5}$$

with base cases

$$S_d(0,0) = S_d(1,1) = 1 \tag{A.6}$$

$$S_d(c,0) = S_d(0,t) = 0 \qquad \text{for } c,t>0 \tag{A.7}$$

$$S_d(c,t) = 0 \qquad \text{for } t>c. \tag{A.8}$$

Another alternative explicit definition is via Toscano's formula [Pitman, 2002, Eq. (3.19)] (see also [Buntine and Hutter, 2012]):

$$S_d(c,t) = \frac{1}{d^t t!} \sum_{j=1}^{t} (-1)^j \binom{t}{j} [-td]_1^n \tag{A.9}$$

Buntine and Hutter [2012] also show that it is possible to directly compute ratios of these Stirling numbers recursively. Such ratios appear in some expressions related to the CRP (e.g. in the probability that a randomly chosen customer sits by himself (4.2)), and being able to compute them directly yields a significant speed up, as it avoids having to compute the Stirling numbers in log space (which is necessary due to their large dynamic range), and also yields more accurate results.

A.3 Posterior Distribution over t_s

Recall the joint distribution over observations/counts and tables (2.33):

$$P(\{c_s,t_s\},z_{1:c}) = \left(\prod_{s\in\Sigma} H(s)^{t_s}\right)\left(\frac{[\alpha+d]_d^{t_\cdot-1}}{[\alpha+1]_1^{c_\cdot-1}} \prod_{s\in\Sigma} S_d(c_s,t_s)\right). \tag{A.10}$$

From this we can see that the (log) posterior distribution over the number of tables t_s is given by

$$\log P(\{t_s\}|\{c_s\},z_{1:c}) = \sum_{s\in\Sigma} \left(t_s \log H(s) + \log S_d(c_s,t_s)\right) + \log[\alpha+d]_d^{t_\cdot-1} + C \tag{A.11}$$

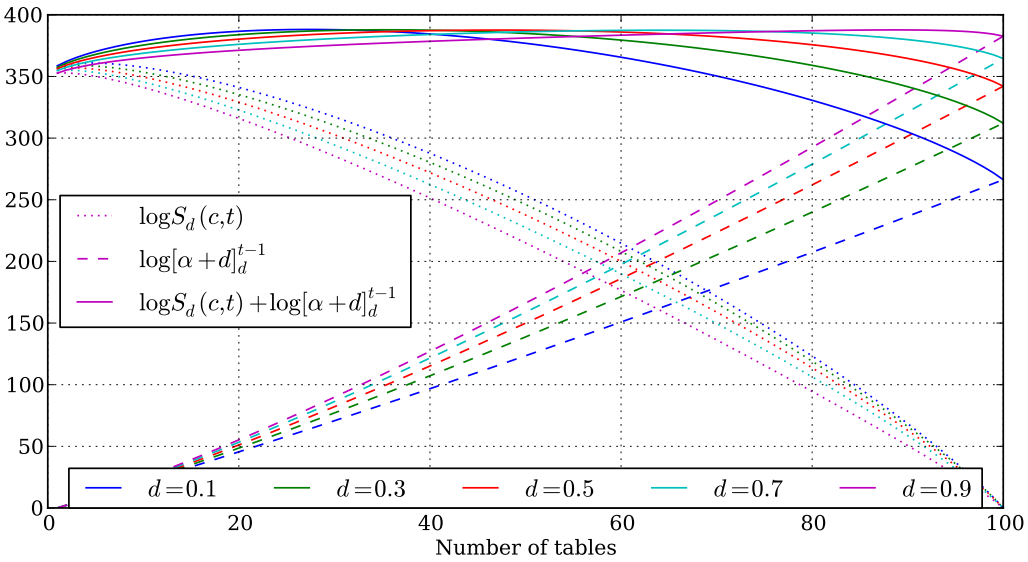


Figure A.1: Components of the (log) CRP posterior distribution of t_s for $c_s = 10$ and $\alpha = 0$.

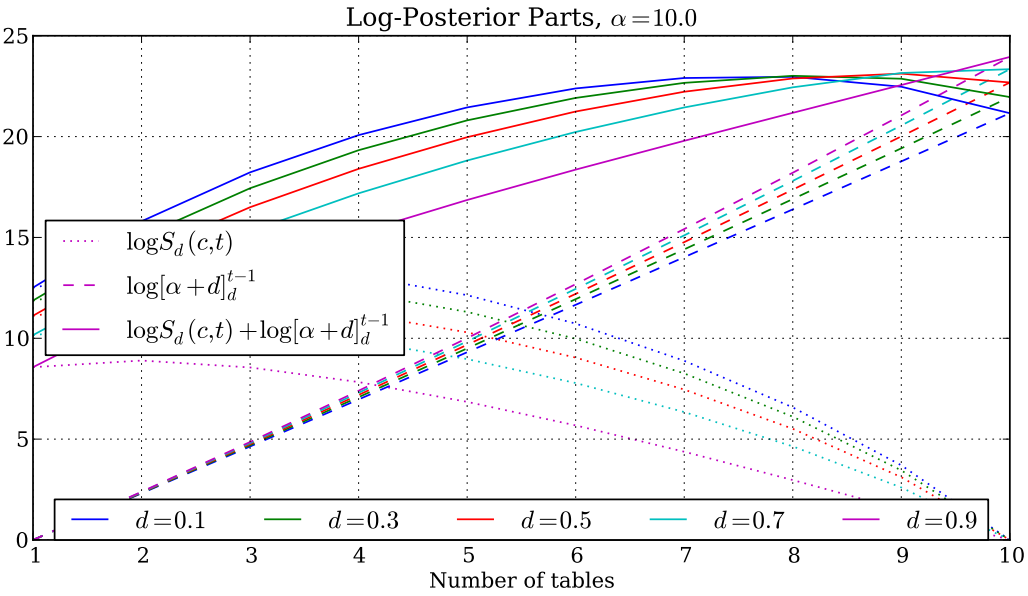


Figure A.2: Components of the (log) CRP posterior distribution of t_s for $c_s = 10$ and $\alpha = 10$.

where C is a constant ensuring normalization. Figures A.1-A.3 shows the individual components of this function, namely $\log S_d(c_s, t_s)$ and $\log[\alpha + d]_d^{t.-1}$ as a function of t_s for varying d (not shown is the linear function $t_s \log G_0$). Figures A.4–A.5 plot the (normalized) posterior distribution of t_s for various values of α, d, and $H(s)$.

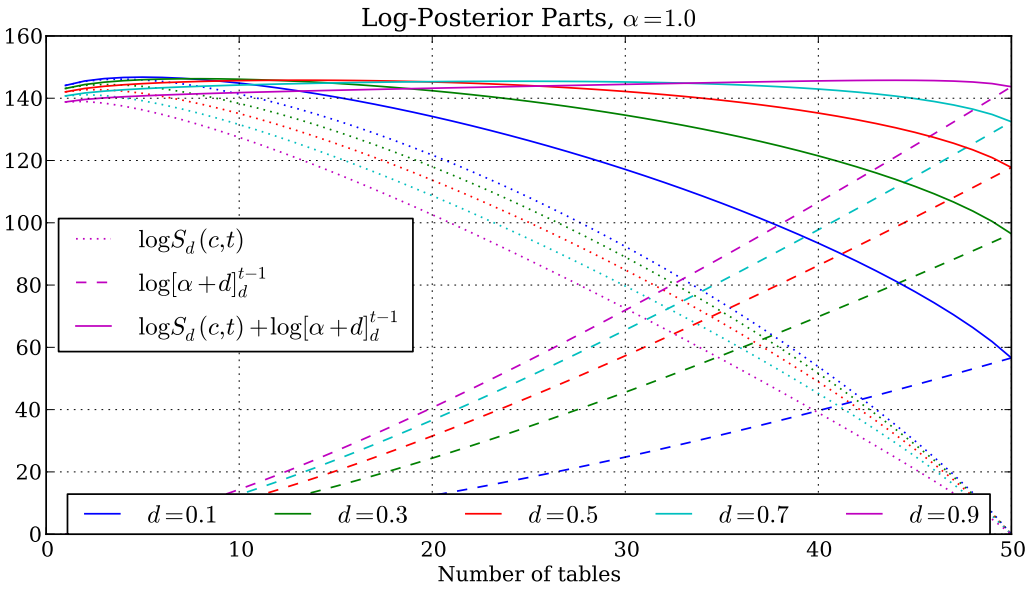


Figure A.3: Components of the (log) CRP posterior distribution of t_s for $c_s = 50$ and $\alpha = 1$.

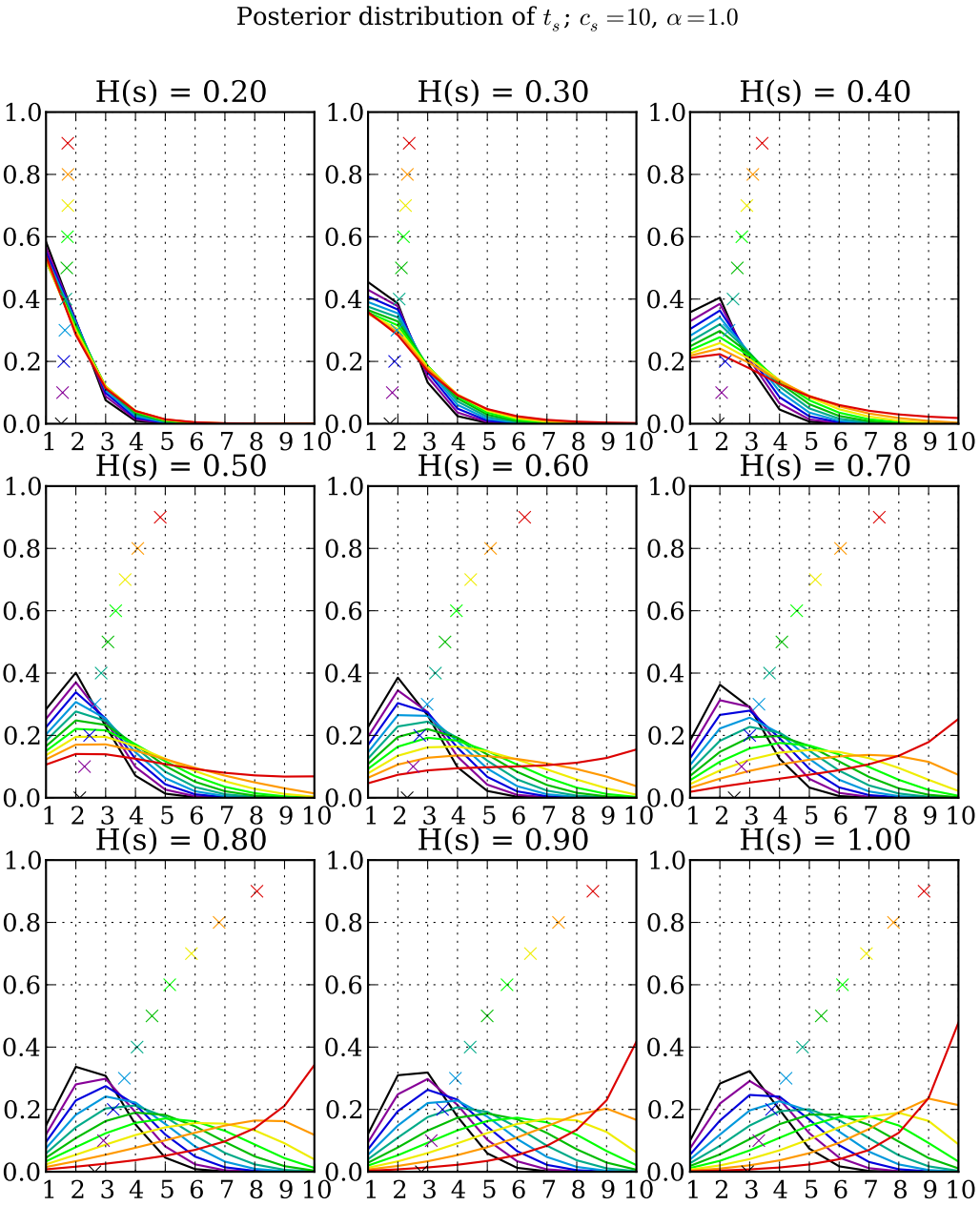


Figure A.4: Posterior distribution over the number of tables t_s in a CRP with $\alpha = 1$ given $c_s = 10$ observations of the same symbol as the base distribution $H(s)$ is varied from 0.2 to 1.0, and the discount parameter d is varied from 0 to 0.9. The crosses mark the points $(E[t_s], d)$ and the lines are drawn in the same color as the corresponding cross. Note that the vertical axis is the posterior mass for the lines, but d for the crosses, both of which are in $[0, 1]$.

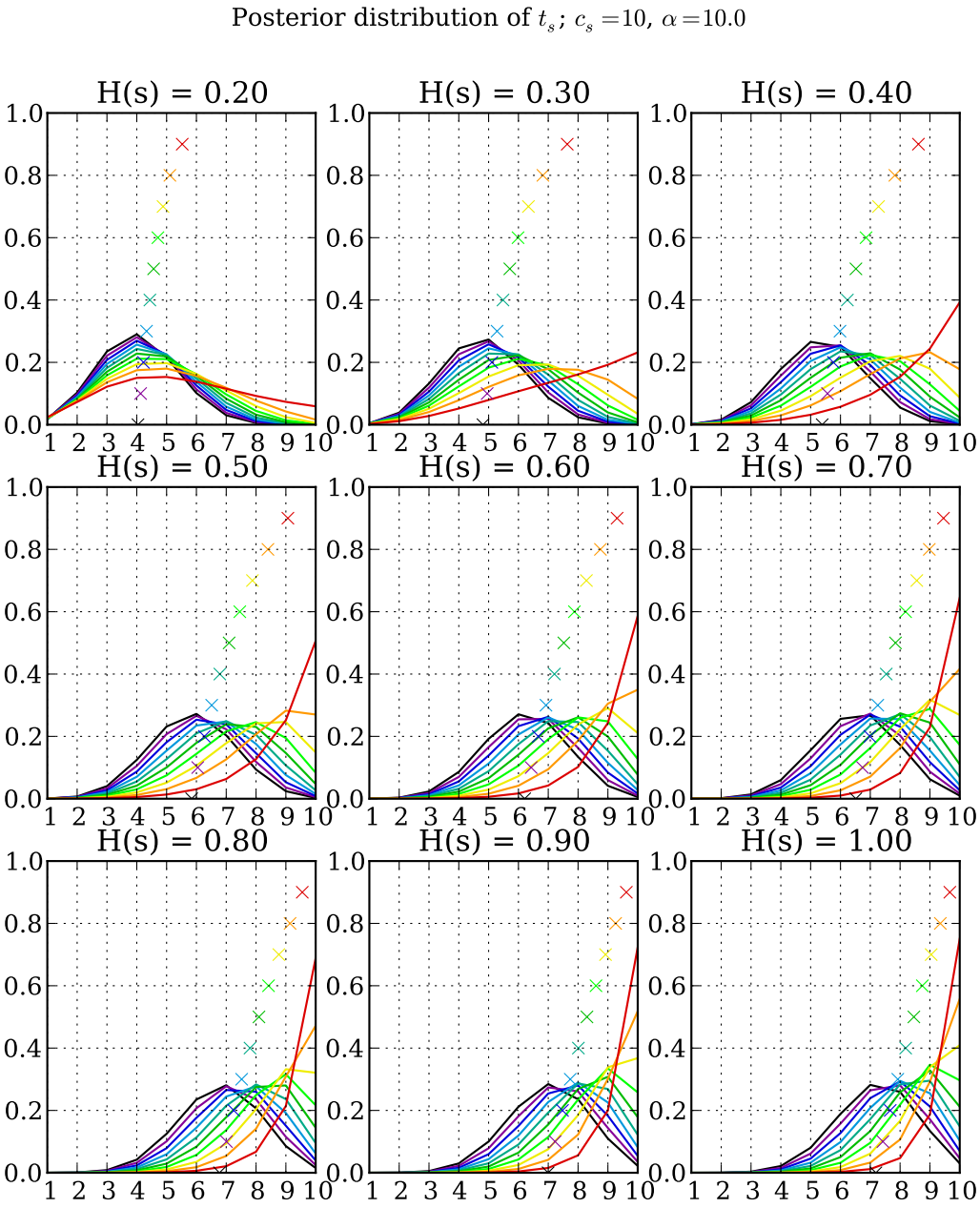


Figure A.5: Same as Figure A.4 but for $\alpha = 10$.

APPENDIX B

Datasets

This appendix provides information on the data sets that were used in this thesis. Most of these data sets have been extensively used by other work on language modelling and compression. Where possible the data sets have been used in the same pre-processed form that has been conventionally used, so that results can be directly compared to previous work. For each data set we provide some background information, information about the preprocessing that has been applied, and descriptive statistics about the data set.

B.1 Penn Treebank (PTB) / Wall Street Journal (WSJ)

n	TR-TYP	VA-TYP	VA-1TYP	VA-1TOK	TE-TYP	TE-1TYP	TE-1TOK
1	10000	6022	0	0	6049	0	0
2	264989	38514	14062	15113	41423	14740	15594
3	615128	61239	42130	43217	67814	46464	47698
4	792627	69064	59343	60014	77339	66963	67809
5	857994	71467	66332	66783	80263	75173	75666
6	883429	72356	69108	69430	81266	78286	78615

Table B.1: n-gram statistics for the WSJ data set. Total number of tokens: 929,589 train; 73,760 validation; 82,430 test. Vocabulary size: 10,000. TR: training set, VA: validation set, TE: test set; TYP: types (unique n-grams), TOK: tokens (number of occurrences), 1TYP and 1TOK: same as TYP and TOK but only counting types/tokens not occurring in training set

Various subsections of the Penn Treebank data set [Marcus et al., 1993] have been used extensively for the evaluation of language modelling techniques. In particular, the Wall Street Journal (WSJ) subsection as originally pre-processed by Bengio et al. [2003] has been used in many comparative stud-

ies involving neural language models, mostly because of its relatively small size (approximately one million tokens) and the small vocabulary size (reduced to 10,000 word types). We refer to this corpus interchangeably as either the Penn Treebank (PTB) or the Wall Street Journal (WSJ) data set. The data set was pre-processed to limit the vocabulary to the 9,998 most frequent words; all occurrences of other words were replaced by a special out-of-vocabulary token. Sentences are delimited by a special sentence end token, so that the total vocabulary size is 10000. The convention for splitting the data set into training, validation, and test set is to use sections 1-20 (929,589 tokens) for training, sections 21-22 (73,760 tokens) for validation, and sections 22-23 (82,430 tokens) for testing. See Table B.1 for n-gram statistics for this data set.

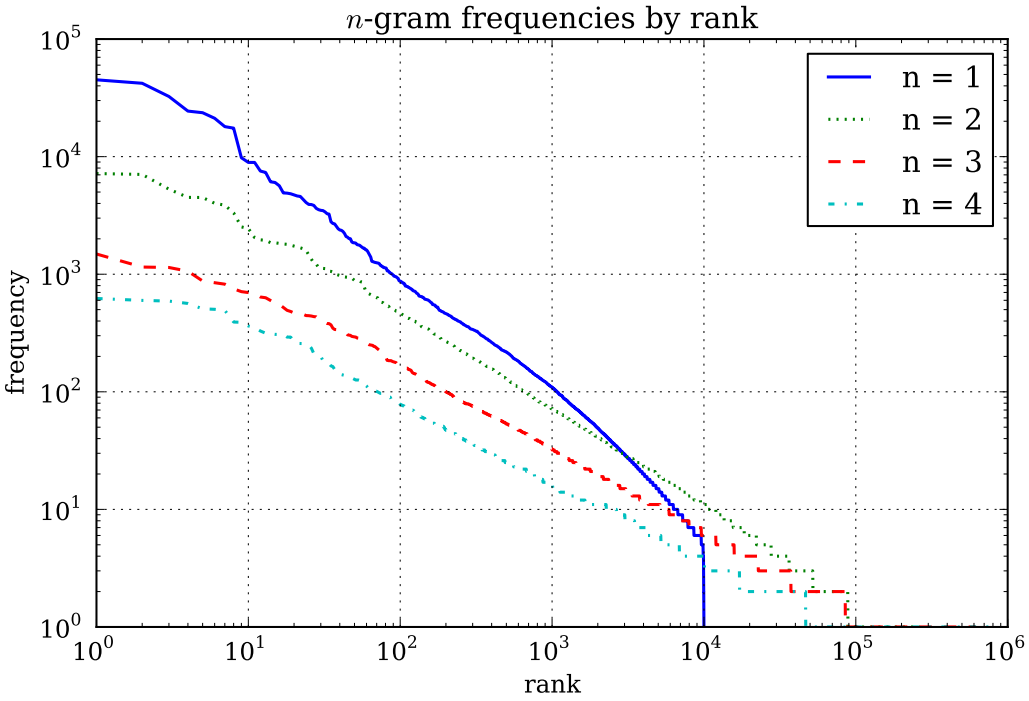


Figure B.1: n-gram frequencies by rank for the PTB/WSJ corpus.

B.2 Brown Corpus

The Brown corpus [Francis and Kucera, 1964] was one of the first and is one of the most widely used corpora of English language text. Here, we use a version that has been preprocessed as in Bengio et al. [2003] and uses the same train/test/validation split that has been used in several other works on language modelling. The training set contains 800,000 tokens, and the validation and test sets contain 200,000 tokens and 181,041 tokens, respectively. The vocabulary contains 16,383 types.

n	TR-TYP	VA-TYP	VA-1TYP	VA-1TOK	TE-TYP	TE-1TYP	TE-1TOK
1	15653	10823	508	2950	9476	448	2505
2	272051	87143	48210	57146	74176	42374	50592
3	592801	161312	132736	141714	142046	118595	127301
4	740459	189529	179982	184840	170237	162663	167116
5	782282	196831	194593	196934	178077	176361	178259
6	792837	198751	198281	199389	180184	179853	180519

Table B.2: n-gram statistics for the Brown corpus dataset. Total number of tokens: 800,000 train; 200,000 validation; 181,041 test; Vocabulary size: 16,383. TR: training set, VA: validation set, TE: test set; TYP: types (unique n-grams), TOK: tokens (number of occurrences), 1TYP and 1TOK: same as TYP and TOK but only counting types/tokens not occurring in training set

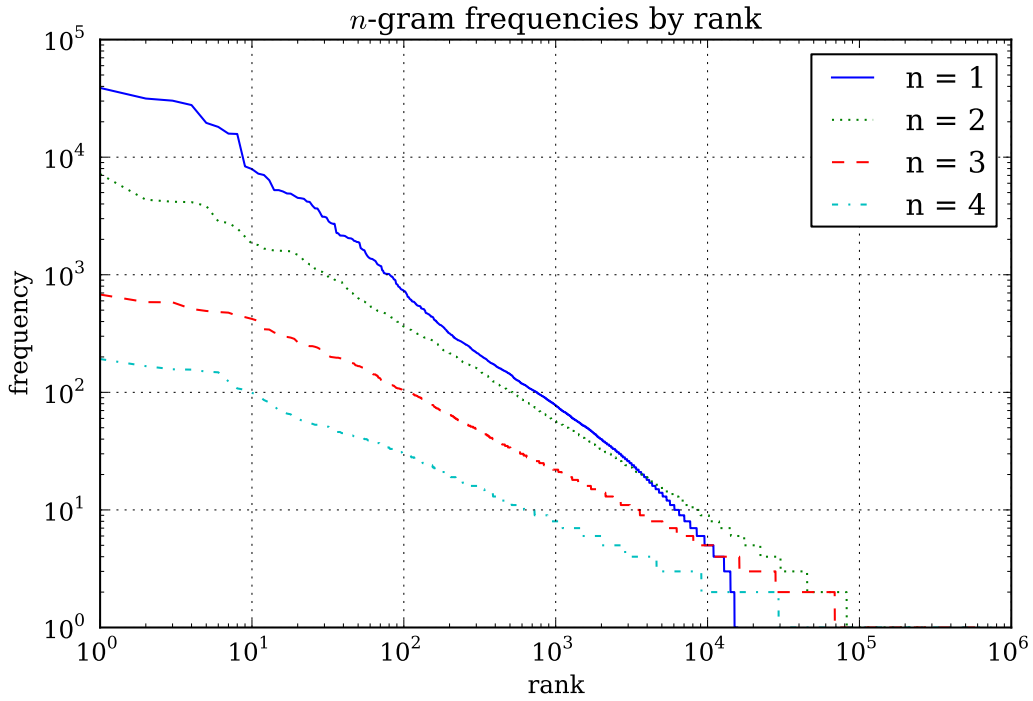


Figure B.2: n-gram frequencies by rank for the Brown Corpus data set.

B.3 AP News Corpus

The AP news corpus is collection of about 16 million tokens of news text from Associated Press collected between 1996 and 1997 [Bengio et al., 2003]. We use the preprocessed version of Bengio et al. [2003] which maps all characters to lower case, and maps numbers, rare words, and proper nouns to special symbols, resulting in a vocabulary of 17,964 tokens. The train/validation/test split is the same as in Bengio et al. [2003] with 13,994,528 training tokens, 963,138 validation tokens, and 964,071 test tokens.

n	TR-TYP	VA-TYP	VA-1TYP	VA-1TOK	TE-TYP	TE-1TYP	TE-1TOK
1	17964	16513	0	0	16597	0	0
2	1461061	255475	62795	73145	255935	63846	74700
3	5349976	572474	304633	344751	575984	309640	351395
4	8875131	752071	571674	633909	757534	582479	646196
5	10751221	825374	728023	795931	830519	740870	810006
6	11606373	856767	800948	868112	861571	814127	881950

Table B.3: n-gram statistics for the AP News dataset. Total number of tokens: 13,994,528 train; 963,138 validation; 963,071 test. Vocabulary size: 17,964. TR: training set, VA: validation set, TE: test set; TYP: types (unique n-grams), TOK: tokens (number of occurrences), 1TYP and 1TOK: same as TYP and TOK but only counting types/tokens not occurring in training set

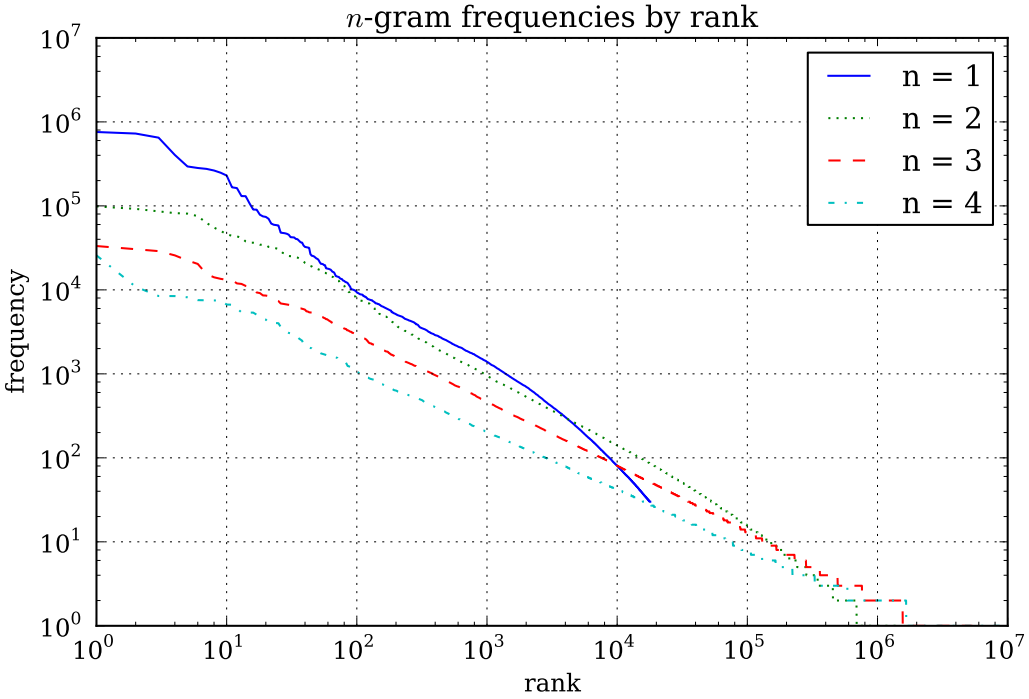


Figure B.3: n-gram frequencies by rank for the AP News corpus.

B.4 Calgary Corpus

The Calgary corpus is a widely-used data compression benchmark data set.[1] It consists of 14 files of various types and sizes with a total size of 3,041,622 bytes (see Table 6.1 for a list of the individual files and sizes). It was originally created researchers at the university of Calgary (I. Witten, T. Bell and J. Cleary).

[1] `http://corpus.canterbury.ac.nz/descriptions/#calgary`

References

Adamic, L. and Huberman, B. (2002). Zipf's law and the Internet. *Glottometrics*, 3:143–150.

Aldous, D. J. (1985). *Exchangeability and related topics*. Springer.

Alumäe, T. and Kurimo, M. (2010). Domain adaptation of maximum entropy language models. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 301–306. Association for Computational Linguistics.

Antoniak, C. E. (1974). Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics*, 2(6):1152–1174.

Apostolico, A., Crochemore, M., Farach-Colton, M., Galil, Z., and Muthukrishnan, S. (2016). 40 years of suffix trees. *Communications of the ACM*, 59(4):66–73.

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

Ayed, F., Lee, J., and Caron, F. (2019). Beyond the Chinese Restaurant and Pitman-Yor processes: Statistical Models with Double Power-law Behavior. *arXiv e-prints*, arXiv:1902.04714.

Bacchiani, M. and Roark, B. (2003). Unsupervised language model adaptation. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP'03).*, volume 1, pages 224–227. IEEE.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*, arXiv:1409.0473.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Bartlett, N., Pfau, D., and Wood, F. (2010). Forgetting counts: Constant memory inference for a dependent hierarchical Pitman-Yor process. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 63–70.

Bartlett, N. and Wood, F. (2011). Deplump for streaming data. In *Data Compression Conference (DCC)*, pages 363–372. IEEE.

Bellegarda, J. R. (2004). Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93–108.

Bellemare, M., Veness, J., and Talvitie, E. (2014). Skip context tree switching. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1458–1466.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Blei, D. M. and Jordan, M. I. (2006). Variational inference for Dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143.

Bloom, C. (1998). Solving the problems of context modeling. `http://www.cbloom.com/papers/ppmz.pdf`.

Blunsom, P. and Cohn, T. (2011). A hierarchical Pitman-Yor process HMM for unsupervised part of speech induction. In *ACL 2011*, pages 865–874.

Blunsom, P., Cohn, T., Goldwater, S., and Johnson, M. (2009). A note on the implementation of hierarchical Dirichlet processes. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 337–340, Suntec, Singapore. Association for Computational Linguistics.

Brants, T. and Franz, A. (2006). *Web 1T 5-gram*. Linguistic Data Consortium, Philadelphia.

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85.

Buntine, W. and Hutter, M. (2012). A Bayesian view of the Poisson-Dirichlet process. *arXiv e-prints*, arXiv:1007.0296v2.

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

Cappé, O., Godsill, S. J., and Moulines, E. (2007). An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924.

Cappé, O. and Moulines, E. (2009). On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613.

Carlton, M. (1999). *Applications of the Two-Parameter Poisson-Dirichlet Distribution*. PhD thesis, UCLA.

Caron, F., Davy, M., and Doucet, A. (2007). Generalized Polya urn for time-varying Dirichlet process mixtures. In *23rd Conference on Uncertainty in Artificial Intelligence (UAI'2007), Vancouver, Canada, July 2007*.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1).

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press.

Chelba, C. and Jelinek, F. (1998). Exploiting syntactic structure for language modeling. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 225–231. Association for Computational Linguistics.

Chelba, C. and Jelinek, F. (2000). Structured language modeling. *Computer Speech & Language*, 14(4):283–332.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., and Koehn, P. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv e-prints*, arXiv:1312.3005.

Chen, C., Du, L., and Buntine, W. (2011). Sampling table configurations for the hierarchical Poisson-Dirichlet process. In *KDD 2011*, pages 296–311.

Chen, S. F. and Goodman, J. T. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–394.

Church, K. W. and Gale, W. A. (1995). Poisson mixtures. *Natural Language Engineering*, 1(2):163–190.

Clarkson, P. R. and Robinson, A. J. (1997). Language model adaptation using mixtures and an exponentially decaying cache. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 799–802. IEEE.

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

Cleary, J. G. and Witten, I. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Collobert, R., Weston, J., and Bottou, L. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Cowans, P. J. (2006). *Probabilistic Document Modelling*. PhD thesis, University of Cambridge.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. *arXiv e-prints*, arXiv:1901.02860.

Dayan, P., Hinton, G., Neal, R. M., and Zemel, R. (1995). Helmholtz machines. *Neural Computation*, 7:1022–1037.

Del Moral, P., Doucet, A., and Singh, S. (2010). Forward smoothing using sequential monte carlo. *arXiv e-prints*, arXiv:1012.5390.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv e-prints*, arXiv:1810.04805.

Douc, R., Cappe, O., and Moulines, E. (2005). Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005*, pages 64–69.

Doucet, A., de Freitas, N., and Gordon, N. J. (2001). *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science. New York: Springer-Verlag.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. (2016). Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Emami, A. and Jelinek, F. (2005). A neural syntactic language model. *Machine learning*, 60(1-3):195–227.

Fearnhead, P. (2004). Particle filters for mixture models with an unknown number of components. *Statistics and Computing*, 14(1):11–21.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, pages 209–230.

Francis, W. N. and Kucera, H. (1964). A standard corpus of present-day edited American english, for use with digital computers. Department of Linguistics, Brown University, Providence, Rhode Island, USA.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29*, pages 1019–1027.

Gasthaus, J. and Teh, Y. W. (2010). Improvements to the Sequence Memoizer. In *Advances in Neural Information Processing Systems 23*, pages 685–693.

Gasthaus, J., Wood, F., Görür, D., and Teh, Y. W. (2009). Dependent Dirichlet process spike sorting. In *Advances in Neural Information Processing Systems 21*, pages 497–504.

Gasthaus, J., Wood, F., and Teh, Y. W. (2010). Lossless compression based on the Sequence Memoizer. In Storer, J. A. and Marcellin, M. W., editors, *Data Compression Conference*, pages 337–345, Los Alamitos, CA, USA. IEEE Computer Society.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian data analysis*. Chapman & Hall/CRC, 2nd edition.

Giegerich, R. and Kurtz, S. (1997). From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353.

Gnedin, A., Hansen, B., and Pitman, J. (2007). Notes on the occupancy problem with infinitely many boxes: general asymptotics and power laws. *Probab. Surveys*, 4:146–171.

Goldwater, S., Griffiths, T., and Johnson, M. (2006a). Interpolating between types and tokens by estimating power-law generators. In *Advances in Neural Information Processing Systems*, volume 18.

Goldwater, S., Griffiths, T., and Johnson, M. (2006b). Interpolating between types and tokens by estimating power-law generators. *Advances in Neural Information Processing Systems 18*, pages 459–466.

Goldwater, S., Griffiths, T., and Johnson, M. (2011). Producing Power-Law Distributions and Damping Word Frequencies with Two-Stage Language Models. *Journal of Machine Learning Research*, 12:2335–2382.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264.

Goodman, J. T. (2001a). A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434.

Goodman, J. T. (2001b). A bit of progress in language modeling. Technical Report MSR-TR-2001-72, Microsoft Research.

Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. (2012). Church: a language for generative models. *arXiv e-prints*, arXiv:1206.3255.

Gough, B. (2009). *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 3rd edition.

Grave, E., Cisse, M. M., and Joulin, A. (2017). Unbounded cache model for online language modeling with open vocabulary. In *Advances in Neural Information Processing Systems*, pages 6042–6052.

Grave, E., Joulin, A., and Usunier, N. (2016). Improving neural language models with a continuous cache. *arXiv e-prints*, arXiv:1612.04426.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE.

Griffin, J. E. (2007). The Ornstein-Uhlenbeck Dirichlet process and other time-varying processes for Bayesian nonparametric inference. Technical report, Department of Statistics, University of Warwick.

Griffin, J. E. and Steel, M. F. J. (2006). Order-based dependent Dirichlet processes. *Journal of the American Statistical Association, Theory and Methods*, 101:179–194.

Griffiths, T. L. and Ghahramani, Z. (2011). The Indian buffet process: An introduction and review. *The Journal of Machine Learning Research*, 12:1185–1224.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Heafield, K., Pouzyrevsky, I., Clark, J., and Koehn, P. (2013). Scalable Modified Kneser-Ney Language Model Estimation. In *ACL 2013*.

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

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.

Ho, M. W., James, L. F., and Lau, J. W. (2006). Coagulation fragmentation laws induced by general coagulations of two-parameter Poisson-Dirichlet processes. *arXiv e-prints*, arXiv:math/0601608.

Hsu, L. C. and Shiue, P. J.-S. (1998). A unified approach to generalized Stirling numbers. *Advances in Applied Mathematics*, 20:366–384.

Huang, S. and Renals, S. (2007). Hierarchical Pitman-Yor language models for ASR in meetings. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, volume 10.

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

Ishwaran, H. and James, L. (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173.

Iyer, R. and Ostendorf, M. (1996). Modeling long distance dependence in language: Topic mixtures vs. dynamic cache models. *IEEE Transactions on Speech and Audio Processing*, pages 236—-239.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam. North-Holland.

Jelinek, F., Merialdo, B., Roukos, S., and Strauss, M. (1991). A dynamic language model for speech recognition. In *HLT*, volume 91, pages 293–295.

Johansson, S. (1985). Word frequency and text type: Some observations based on the LOB corpus of British English texts. *Computers and the Humanities*, 19(1):23–36.

Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv e-prints*, arXiv:1602.02410.

Katz, S. M. (1996). Distribution of content words and phrases in text and language modelling. *Natural Language Engineering*, 2(1):15–59.

Kingman, J. (1975). Random discrete distributions. *Journal of the Royal Statistical Society. Series B*, 37(1):1–22.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184.

Kneser, R. and Steinbiss, V. (1993). On the dynamic adaptation of stochastic language models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 21, pages 586–589.

Knoll, B. (2011). A machine learning perspective on predictive coding with PAQ8 and new applications. Master's thesis, University of British Columbia.

Knoll, B. and de Freitas, N. D. (2012). A machine learning perspective on predictive coding with PAQ8. In *Data Compression Conference*, pages 377–386.

Kong, A., Liu, J. S., and Wong, W. H. (1994). Sequential imputations and Bayesian missing data problems. *Journal of the American Statistical Association*, 89(425):278–288.

Krause, B., Kahembwe, E., Murray, I., and Renals, S. (2017). Dynamic evaluation of neural sequence models. *arXiv e-prints*, arXiv:1709.07432.

Krause, B., Kahembwe, E., Murray, I., and Renals, S. (2019). Dynamic evaluation of transformer language models. *arXiv e-prints*, arXiv:1904.08378.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105.

Kuhn, R. (1988). Speech recognition and the frequency of recently used words: A modified Markov model for natural language. In *Proceedings of the 12th conference on Computational linguistics-Volume 1*, pages 348–350. Association for Computational Linguistics.

Kuhn, R. and De Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(6):570–583.

Lucas-Cuesta, J., Ferreiros, J., Fernández-Martinez, F., Echeverry, J., and Lutfi, S. (2013). On the dynamic adaptation of language models based on dialogue information. *Expert Systems with Applications*, 40(4):1069 – 1085.

MacEachern, S. N. (1999). Dependent nonparametric processes. In *Proceedings of the Section on Bayesian Statistical Science*. American Statistical Association.

MacEachern, S. N. (2000). Dependent Dirichlet processes. Technical report, Department of Statistics, Ohio State University.

MacKay, D. J. C. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

MacKay, D. J. C. and Bauman Peto, L. (1995). A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(2):289–307.

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

Mahoney, M. (2013). Data compression explained. URL: `http://mattmahoney.net/dc/dce.html`.

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

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Meka, R., Jain, P., and Dhillon, I. S. (2009). Matrix completion from power-law distributed samples. In *Advances in Neural Information Processing Systems 22*, pages 1258–1266.

Melis, G., Dyer, C., and Blunsom, P. (2017). On the state of the art of evaluation in neural language models. *arXiv e-prints*, arXiv:1707.05589.

Merity, S., Keskar, N. S., and Socher, R. (2017). Regularizing and optimizing LSTM language models. *arXiv e-prints*, arXiv:1708.02182.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *arXiv e-prints*, arXiv:1609.07843.

Metropolis, A. W., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092.

Miikkulainen, R. and Dyer, M. G. (1991). Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15(3):343–399.

Mikolov, T. (2012). *Statistical Language Models Based On Neural Networks*. PhD thesis, Brno University of Technology.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Černockỳ, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth Annual Conference of the International Speech Communication Association*.

Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. *SLT*, 12:234–239.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, pages 641–648.

Mnih, A. and Hinton, G. (2009). A scalable hierarchical distributed language model. In *Neural Information Processing Systems 22*.

Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, pages 1751–1758.

Mnih, A., Yuecheng, Z., and Hinton, G. (2009). Improving a statistical language model through non-linear prediction. *Neurocomputing*, 72(7-9):1414 – 1418.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv e-prints*, arXiv:1312.5602.

Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921.

Montavon, G., Orr, G., and Müller, K.-R., editors (2012). *Neural Networks: tricks of the trade*. Springer, 2nd edition.

Neal, R. M. (1992). Bayesian mixture modeling. In *Proceedings of the Workshop on Maximum Entropy and Bayesian Methods of Statistical Analysis*, volume 11, pages 197–211.

Neal, R. M. (1998). Markov chain sampling methods for Dirichlet process mixture models. Technical Report 9815, Department of Statistics, University of Toronto.

Neal, R. M. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9:249–265.

Nelson, M. (2014). Data compression with arithmetic coding. `http://marknelson.us/2014/10/19/data-compression-with-\ arithmetic-coding/`.

Neubig, G. and Dyer, C. (2016). Generalizing and hybridizing count-based and neural language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Austin, Texas, USA.

Newman, M. (2005). Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351.

Ney, H., Essen, U., and Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer.

Pasco, R. C. (1976). *Source coding algorithms for fast data compression*. PhD thesis, Stanford University.

Pauls, A. and Klein, D. (2011). Faster and smaller n-gram language models. In *ACL 2011*.

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

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.

Piantadosi, S. T. (2014). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, 21(5):1112–1130.

Pitman, J. (1992). The two-parameter generalization of Ewens' random partition structure. Technical Report 345, Department of Statistics, U.C. Berkeley.

Pitman, J. (1995). Exchangeable and partially exchangeable random partitions. *Probability Theory and Related Fields*, 102(2):145–158.

Pitman, J. (1996). Some developments of the Blackwell-MacQueen urn scheme. In Ferguson, T. S., Shapley, L. S., and MacQueen, J. B., editors, *Statistics Probability and Game Theory; Papers in honor of David Blackwell*, Lecture Notes – Monograph Series, pages 245–267. Institute of Mathematical Statistics.

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. Lecture notes for St. Flour Summer School.

Pitman, J. and Yor, M. (1997). The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2):855–900.

Popescu-Belis, A. (2019). Context in neural machine translation: A review of models and evaluations. *arXiv e-prints*, arXiv:1901.09115.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rao, V. and Teh, Y. W. (2009). Spatial normalized gamma processes. In *Advances in Neural Information Processing Systems*, volume 22, pages 1554–1562.

Rao, V. and Teh, Y. W. (2013). Fast MCMC sampling for Markov jump processes and extensions. *The Journal of Machine Learning Research*, 14(1):3295–3320.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Rissanen, J. (1976). Generalized kraft inequality and arithmetic coding. *IBM Journal of research and development*, 20(3):198–203.

Rissanen, J. and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162.

Roark, B. (2001). Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Rosenfeld, R. (1994). *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. PhD thesis, Carnegie Mellon University.

Shareghi, E. (2017). *Scalable Non-Markovian Sequential Modelling for Natural Language Processing*. PhD thesis, Monash University.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Srebro, N. and Roweis, S. (2005). Time-varying topic models using dependent Dirichlet processes. Technical Report UTML-TR-2005-003, Department of Computer Science, University of Toronto.

Steinruecken, C. (2014). *Lossless Data Compression*. PhD thesis, University of Cambridge.

Sudderth, E. B. and Jordan, M. I. (2009). Shared segmentation of natural scenes using dependent pitman-yor processes. In *Neural Information Processing Systems 22*, pages 1585–1592.

Takahashi, S. and Tanaka-Ishii, K. (2017). Do neural nets learn statistical laws behind natural language? *PLOS ONE*, 12(12):1–17.

Teh, Y. W. (2006a). A Bayesian interpretation of interpolated Kneser-Ney. Technical Report TRA2/06, School of Computing, National University of Singapore.

Teh, Y. W. (2006b). A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 985–992.

Teh, Y. W. (2010). Dirichlet processes. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning*. Springer.

Teh, Y. W. and Jordan, M. I. (2010). Hierarchical Bayesian nonparametric models with applications. In Hjort, N., Holmes, C., Müller, P., and Walker, S., editors, *Bayesian Nonparametrics*. Cambridge University Press.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2004). Hierarchical Dirichlet processes. Technical Report 653, Department of Statistics, University of California at Berkeley.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2005). Sharing clusters among related groups: Hierarchical Dirichlet processes. In *Advances in Neural Information Processing Systems*, volume 17.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

Tiedemann, J. (2010). Context adaptation in statistical machine translation using models with exponentially decaying cache. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 8–15.

Tran, D., Hoffman, M. D., Saurous, R. A., Brevdo, E., Murphy, K., and Blei, D. M. (2017). Deep probabilistic programming. In *International Conference on Learning Representations*.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14:249–260.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv e-prints*, arxiv:1706.03762.

Veness, J., Ng, K. S., Hutter, M., and Bowling, M. (2012). Context tree switching. In *Data Compression Conference (DCC), 2012*, pages 327–336. IEEE.

Weiner, P. (1973). Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11.

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/`.

Willems, F. M. J., Shtarkov, Y. M., and Tjalkens, T. J. (1995). The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664.

Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *Information Theory, IEEE Transactions on*, 37(4):1085–1094.

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. F., and Teh, Y. W. (2009). A stochastic memoizer for sequence data. In *Proceedings of the International Conference on Machine Learning*, volume 26, pages 1129–1136.

Wood, F., Gasthaus, J., Archambeau, C., James, L., and Teh, Y. W. (2011). The sequence memoizer. *Communications of the Association for Computing Machines*, 54(2):91–98.

Wood, F. and Teh, Y. W. (2009). A hierarchical nonparametric Bayesian approach to statistical language model domain adaptation. In *JMLR Workshop and Conference Proceedings: AISTATS 2009*, volume 5, pages 607–614.

Wood, F., van de Meent, J. W., and Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, pages 1024–1032.

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

Xu, P. and Jelinek, F. (2004). Random forests in language modeling. In *Proc. EMNLP*.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent Neural Network Regularization. *arXiv e-prints*, arXiv:1409.2329.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2017). Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4189–4198.