A Particle Filter algorithm for Bayesian Wordsegmentation

Benjamin Börschinger
Department of Computing
Macquarie University
Sydney
benjamin.borschinger@mq.edu.au

Mark Johnson
Department of Computing
Macquarie University
Sydney
mark.johnson@mq.edu.au

Abstract

Bayesian models are usually learned using batch algorithms that have to iterate multiple times over the full dataset. This is both computationally expensive and, from a cognitive point of view, highly implausible. We present a novel online algorithm for the word segmentation models of Goldwater et al. (2009) which is, to our knowledge, the first published version of a Particle Filter for this kind of model. Also, in contrast to other proposed algorithms, it comes with a theoretical guarantee of optimality if the number of particles goes to infinity. While this is, of course, a theoretical point, a first experimental evaluation of our algorithm shows that, as predicted, its performance improves with the use of more particles, and that it performs competitively with other online learners proposed in Pearl et al. (2011).[1]

1 Introduction

Bayesian models have recently become quite popular in Computational Linguistics. One undesirable property of many such models is, however, that the inference algorithms usually applied to them, in particular popular Markov Chain Monte Carlo Methods such as Gibbs Sampling, require multiple iterations over the data — this is both computationally expensive and, from a cognitive point of view, implausible. Online learning algorithms directly address this problem by requiring only a single pass over the data, thus providing a first step from 'ideal learner' analyses towards more realistic learning scenarios (Pearl et al., 2011). In this paper, we present a Particle Filter algorithm for the word segmentation models of Goldwater et al. (2009), to our knowledge showing for the first time how a Particle Filter can be applied to models of this kind. Notably, Particle Filters are mathematically well-motivated algorithms to produce a finite approximation to the true posterior of a model, the quality of which increases with larger numbers of particles and recovering the true posterior if the number of particles goes to infinity. This sets them qualitatively apart from most previously proposed online learners that usually are based on heuristic ideas.

The structure of the rest of the paper is as follows. First, we give a high-level description of the Bayesian word segmentation model our algorithm applies to and make explicit our notion of an online learner. Then, we give a quick overview of previous work and go on to describe the word segmentation model in more detail, introducing the relevant notation and formulae. Finally, we give a description of the algorithm and present experimental results, comparing the algorithm with other proposed learning algorithms for the model and its performance across different numbers of particles.

2 The Goldwater model for word segmentation

The model[2] assumes that a segmented text is created by a random process that generates a sequence

[1]The source code for our implementation is available for download from `http://web.science.mq.edu.au/~bborschi/`

[2]We only provide a high-level idea of the Bayesian Unigram model for word segmentation of Goldwater et al. (2009). For more details and a description of the Bigram model, we refer the reader to the original paper.

of words $\sigma = w_{1:n}$ which can be interpreted as a segmentation of the unsegmented text T that is the result of concatenating these words.[3]

The first word is generated by a distribution over possible words, the so-called base distribution P_0 that, in principle, can generate words of unbounded length. We'll come back to the details of the base distribution in section 4.1. Each further word is either generated by 'reusing' one of the previously generated words, or by making a new draw from the base distribution. This generative process, also known as the (labelled) Chinese Restaurant Process, is formally described as:

$$P(W_1{=}w \mid \alpha) = P_0(w) \tag{1}$$

$$P(W_{i+1}{=}w \mid W_{1:i}, \alpha) = \frac{c_w(W_{1:i}) + \alpha P_0(w)}{i + \alpha} \tag{2}$$

where P_0 is the base distribution over words and $c_w(W_{1:i})$ is the number of times the word w has been observed in the sequence $W_{1:i}$. α is the hyperparameter for the process, also known as the concentration parameter, and controls the probability of generating previously unseen words by making a new draw from P_0.

This process can be understood in terms of a restaurant metaphor: each generated word corresponds to the order of a customer in a restaurant, and each customer who enters the restaurant either sits at an already occupied table with probability proportional to the number of people already sitting there, ordering the exact same dish they are already eating (the label of the table), or sits at a new table with probability proportional to α and orders a new dish which corresponds to making a draw from the base distribution.[4] In principle, there is an infinite number of tables in the restaurant but we only are interested in those that are actually occupied, allowing us to actually represent the state of this process with finite means. We will be using the metaphor of customers and tables in the following, for ease of presentation and lack of a better terminology.

The conditional distributions defined by eq. 1 and eq. 2 is exchangeable, i.e. every permutation of the same sequence of words is assigned the same probability. This allows us to completely capture the state of the generative process after having generated i words by a description of the seating for the i customers.

2.1 Inference for the model

While this description has focused on the generative side of the model, probabilistic models like this are usually not used to *generate* random sequences of words but to do *inference*. In this case, we are interested in the posterior distribution over segmentations σ for a specific text T, $P(\sigma \mid T)$.

While it is easy to calculate the probability of any given segmentation using eq. 1 and eq. 2, determining the posterior distribution or even just finding the most probable segmentation is computationally intractable. Instead, an approximation to the posterior can be calculated using, for example, Markov Chain Monte Carlo algorithms such as Gibbs Sampling. Going into the details of Gibbs Sampling is beyond the scope of the paper, and in fact we propose an alternative algorithm here. We refer the reader to the original Goldwater et al. (2009) paper for a detailed description of their Gibbs Sampling algorithm and to Bishop (2006) for a general introduction.

2.2 Motivation for Online Algorithms

Gibbs Sampling is a batch method that requires multiple iterations over the whole data — in practice, it is not uncommon to have 20,000 iterations on the amount of data we are working with here. This is both computationally expensive and, from a cognitive point of view, highly implausible. Having online learning algorithms is therefore a desirable goal, and their failure to obtain an optimal solution can be seen as telling us how a constrained learner might make use of certain models; in this sense, they provide a first step from ideal learner analyses to more realistic settings (Pearl et al., 2011).

Constraints on Online Algorithms In our opinion, a plausible constraint on an online learner is that it (a) sees each example only once[5] and (b) has to

[3]We take an expression of the form $x_{1:n}$ to refer to the sequence $x_1, \ldots, x_n$.

[4]Note that the same word may label several different tables, as the base distribution may generate the same word multiple times.

[5]Note that this applies to example tokens. There may well be multiple tokens of the same example type.

make a learning decision on the basis of one example at a time immediately after having seen it, using a finite amount of computation. While this is certainly a very strict view, we think it is a plausible first approximation to the constraints human learners are subject to, and it is certainly interesting in and of itself to see how well a learner constrained in this manner can perform. It will be an interesting question for future research to see how relaxing these constraints to a certain extent effects the performance of the learner.

Note that our constraints on online learners exclude certain algorithms that have been labelled 'online' in the literature. For example, the Online EM algorithms in Liang and Klein (2009) make local updates but iterate over the whole data multiple times, thus violating (a). Pearl et al.'s (2011) DMCMC algorithm, discussed in the next section, is able to revisit earlier examples in the light of new evidence, violating thus both (a) and (b).

3 Previous work

Online learning algorithms for Bayesian models are discussed within both Statistics and Computational Linguistics but have, to our knowledge, not yet been widely applied to the specific problem of word segmentation. Both Brent (1999) and Venkataraman (2001) propose heuristic online learning algorithms employing Dynamic Programming that have, however, been shown to not actually maximize the objective defined by the model (Goldwater, 2007). Brent's algorithm has recently been reconsidered as an online learning algorithm in Pearl et al. (2011).

It is an instance of the familiar Viterbi algorithm that efficiently finds the optimal solution to many problems; not, however, for the word segmentation models under discussion. The algorithm is "greedy" in that it determines the locally optimal segmentation for an utterance given its current knowledge using Dynamic Programming, adding the words of this segmentation to its knowledge and proceeding to the next utterance. Both this Dynamic Programming Maximization (DPM) algorithm and a slight variant called Dynamic Programming Sampling (DPS) are described in detail in Pearl et al., the main difference between the two algorithms being that the latter does not pick the most probable segmentation but rather samples a segmentation according to its probability. Note that DPS is, in effect, a Particle Filter with just one particle.

Pearl et al. also present a Decayed Markov Chain Monte Carlo algorithm (Marthi et al., 2002) that is basically an 'online' version of Gibbs Sampling. For each observed utterance, the learner is allowed to reconsider any possible boundary position it has encountered so far in light of its current knowledge, but the probability of reconsidering any specific boundary position decreases with its distance from the current utterance. In effect, boundaries in recent utterances are more likely to be reconsidered than boundaries in earlier ones. While this property is nice in that it can be interpreted as some kind of memory decay, the algorithm breaks our constraints on online learners, as has already been mentioned above: the DMCMC learner explicitly remembers each training example, effectively giving it the ability to learn from and see each example multiple times. It has, in a sense, "knowledge of 'future' utterances when it samples boundaries further back in the corpus than the current utterance", as Pearl et al. point out themselves.

As for non-online algorithms, the state of the art for this word segmentation problem is the adaptor grammar MCMC sampler of Johnson and Goldwater (2009), which achieves 87% word token f-score on the same test corpus as used here. The adaptor grammar model learns both syllable structure and inter-word dependencies, and performs Bayesian inference for the optimal hyperparameters and word segmentation.

4 Model Details

Our models are basically the unigram and bigram models described in Goldwater et al. (2009) and quickly introduced above. There is, however, an important difference with respect to the choice of the base distribution that we describe in what follows. Also, our assumption about what constitutes a hypothesis is different from Goldwater et al., which is why we describe it in some detail.[6] The mathematical details are given in figure 1 while in the text, we focus on a high-level explanation of the ideas.

[6]Again, we focus on the Unigram model.

$$P_c(C = k \mid s, \phi) = \frac{cc_{s,k} + \phi}{\sum_j cc_{s,j} + |\text{chars}|\phi} \quad (j \in \text{chars}) \tag{3}$$

$$P_0(W = k_{1:n} \mid s, \phi) = \left(\prod_{i=1}^{n} P_c(k_i \mid s, \phi) \right) \times P_c(\# \mid s, \phi) \tag{4}$$

$$P_{\text{add}}(\langle wo, t \rangle \mid s, \alpha) = \begin{cases} \frac{ct_{s,t}}{ct_{s,\cdot} + \alpha} & \text{if } t \in s \text{ and } wo \text{ is the label of } t \\ \frac{\alpha P_0(wo \mid \text{labels}(s), \phi)}{ct_{s,\cdot} + \alpha} & \text{if } t \text{ is a new table.} \end{cases} \tag{5}$$

$$P_\sigma(\boldsymbol{\sigma} \mid s, \alpha) = P_{\text{add}}(\sigma_1 \mid s, \alpha) \times \cdots \times P_{\text{add}}(\sigma_n \mid s \cup \sigma_1 \cup \cdots \cup \sigma_{n-1}) \tag{6}$$

$$Q_\sigma(\boldsymbol{\sigma} \mid s, \alpha) = \prod_{i=1}^{n} P_{\text{add}}(\sigma_i \mid s, \alpha) \tag{7}$$

$$\boldsymbol{\sigma}_{i+1}^{(l)} \sim Q_\sigma(\cdot \mid s_i^{(l)}, o_{i+1}, \alpha) \tag{8}$$

$$s_{i+1}^{(l)} = s_i^{(l)} \bigcup \boldsymbol{\sigma}_{i+1}^{(l)} \tag{9}$$

$$\begin{aligned} w_{i+1}^{*(l)} &= w_i^{(l)} \frac{P(o_{i+1} \mid s_{i+1}^{(l)}, \alpha) P(s_{i+1}^{(l)} \mid s_i^{(l)}, \alpha)}{Q(s_{i+1}^{(l)} \mid s_i^{(l)}, o_{i+1}, \alpha)} = w_i^{(l)} \frac{P(o_{i+1}, s_{i+1}^{(l)} \mid s_i^{(l)}, \alpha)}{Q_\sigma(\boldsymbol{\sigma}_{i+1}^{(l)} \mid s_i^{(l)}, o_{i+1}, \alpha)} \\ &= w_i^{(l)} \frac{P_\sigma(\boldsymbol{\sigma}_{i+1}^{(l)} \mid s_i^{(l)}, \alpha)}{Q_\sigma(\boldsymbol{\sigma}_{i+1}^{(l)} \mid s_i^{(l)}, o_{i+1}, \alpha)} \end{aligned} \tag{10}$$

$$w_{i+1}^{(l)} = \frac{w_{i+1}^{*(l)}}{\sum_{j=1}^{N} w_{i+1}^{*(j)}} \tag{11}$$

$$\widehat{ESS_i} = \frac{1}{\sum_j^N (w_i^{(j)})^2)} \tag{12}$$

Figure 1: The mathematical details needed for implementing the algorithm. $cc_{s,k}$ is the number of times character k has been observed in the words in $\text{labels}(s)$ which, in turn, refers to the words labeling (unigram) tables in model state s. chars is the set of different characters in the language, including the word-boundary symbol $\#$. $ct_{s,t}$ refers to the number of customers at table t in model state s, and $ct_{s,\cdot}$ refers to the total number of customers in s. A segmentation $\boldsymbol{\sigma}$ is a sequence of word-table pairs $\langle wo, t \rangle$ that indicates both which words make up the segmentation and at which table each word customer is seated. $s \cup \sigma_i$ refers to the model state that results from adding the i^{th} word-table pair of $\boldsymbol{\sigma}$ to hypothesis s, and the probability of adding this pair is given by P_{add} in eq. 5. $s \bigcup \boldsymbol{\sigma}$ refers to adding all word-table pairs in $\boldsymbol{\sigma}$ to s. Q_σ is the proposal distribution from which we can efficiently sample segmentations, given an observation o, i.e. an unsegmented utterance, and a model state s. P_σ is the true distribution over segmentations according to which we can efficiently score proposals to calculate (the unnormalized) weights w^* using eq. 10. Eq. 8 and eq. 10 can be calculated because $Q_\sigma(\boldsymbol{\sigma} \mid s, \alpha, o) = \frac{Q(\boldsymbol{\sigma}, o \mid s, \alpha)}{Q(o \mid s, \alpha)} = \frac{Q_\sigma(\boldsymbol{\sigma} \mid s, \alpha)}{Q(o \mid s, \alpha)}$, the denominator of which can be efficiently calculated using the forward-algorithm.

4.1 The Lexical-Model

Even though Goldwater et al. found the choice of the lexical model to make virtually no difference for the performance of an unconstrained learner, this does not hold for online learners, an observation already made by Venkataraman (2001). In the original model, each character is assumed to have the same probability $\theta_0 = \frac{1}{|characters|}$ of being generated, and words are generated by a zero-order (Unigram) markov process with a fixed stopping probability. In contrast, we assume that there is a symmetric Dirichlet prior with parameter ϕ on the probability distribution over characters:

$\boldsymbol{\theta} \sim Dir(\phi)$

$P_c(k) = \theta_k$

By integrating out θ, we get a 'learned' conditional distribution over characters, and consequently words, given the learner's lexicon up to that point (eq. 3 and eq. 4).

In addition, we do not fix the probability for the word-boundary symbol, treating it as a further special character $\#$ that may only occur at the end of a word.[7]

[7]While this makes possible in principle the generation of empty words, we are confident that this does not pose a prac-

4.2 Probability of a segmentation

Since we are interested in the posterior distribution over hypotheses given unsegmented utterances, it is important to be clear about what constitues a hypothesis. At a high level, a hypothesis s can be thought of as a lexicon that arises from the segmentation decisions made for the observations up to this point. For example, if the learner previously assumed the segmentations “a dog” and “a woman”, its lexicon contains two occurences of “a” and one occurence of “dog” and “woman”, respectively.

More precisely, a hypothesis is a model state and a model state is an assignment of observed (or rather, previously predicted) word ‘customers’ to tables (see section 2).[8] At time k, the model state s_k is the seating arrangement after having segmented the current observation o_k given the previous model state s_{k-1}, where each observation is an unsegmented string. As our incremental learner can only make additive changes to the ‘restaurant’ — no customers ever leave the table they are assigned to — the hypothesis at time $k+1$ is uniquely determined by the seating arrangement at time k and the proposed segmentation for observation o_{k+1}, as a segmentation not only indicates the actual words, e.g. “a” and “dog”, but also the table at which each word should be seated (which may be a new table). Thus, going from one model state to the next corresponds to sampling a segmentation for the new observation (eq. 8) and adding this segmentation to the current model state (eq. 9). The formulae that assign probabilities to segmentations are given in eq. 5 to eq. 7.

5 The Particle Filter algorithm

Our algorithm is an instance of a Particle Filter, or more precisely, of the Sequential Importance Resampling (SIR) algorithm (Doucet et al., 2000). The idea is to sequentially approximate a target posterior distribution P by N weighted point samples or particles, updating each particle and its weight in light of each succeeding observation. Hypotheses that do conform to the data gain weight, mimicking a kind of a “survival of the fittest” dynamic. Notably, the accuracy of the approximation increases with the number of particles, a result borne out by our experiments.

At a very high-level, a Particle Filter is very similar to a stochastic beam-search in that a number of possibilities is explored in parallel, and the choice of possibilities that are further explored is biased towards locally good, i.e. high-probable ones.

At any given stage, the weighted particles give a finite approximation to the target posterior distribution up to that point, the weight of each particle representing its probability under the approximation. Therefore, we can use it to approximate any expectation of the posterior, e.g. some measure of its word segmentation accuracy, as a weighted sum.

5.1 Description of the Algorithm

A formal description of the algorithm is given in Figure 2 which we explain in more detail here. The algorithm starts at time $i = 0$ with N identical model states (particles) $s_0^{(n)}$, in our case empty lexicons, or rather empty restaurants.[9] At each time step $i+1$, it propagates each particle by sampling a segmentation $\sigma_{i+1}^{(l)}$ for the next observation o_{i+1} according to the the current model state $s_i^{(l)}$ (eq. 8) and adding this segmentation to it, yielding the updated particle $s_{i+1}^{(l)}$ (eq. 9). Intuitively, at each step the learner predicts a segmentation for the current observation in light of what it has learned so far. Adding a segmentation to a model state corresponds to adding the word customers in the segmentation to the corresponding tables. As there are multiple particles, in principle the algorithm can explore alternative hypotheses, reminiscent of a beam-search.

Not all hypotheses, however, fit the observations equally well, and as new data becomes available at each time step, the relative merit of different hypotheses may change. This is captured in a particle filter by assigning weights $w_i^{(l)}$ to each particle that are iteratively updated, using eq. 10 and eq. 11. This update takes both into account how well the particle fit previously seen data in the form of the old weight, and how well it fits the last observation in the form

tical problem as we only use the model for inference, not for generation.

[8]The reason our description involves an explicit record of table assignments is that this is needed for the Bigram model. While actually not needed for the Unigram case, our formulation can be extended to the Bigram model in a straight-forward way, given the description in Goldwater et al. (2009).

[9]The superscript indexes the individual particles, the subscript indicates the time.

of the probability of the proposed segmentation under the current model.

Also, the formula we use for calculating the particle weights, taken from Doucet et al. (2000), overcomes a fundamental problem in applying Particle Filters to this kind of model: we are usually not able to efficiently sample directly from P, because P does not decompose in the way required for Dynamic Programming (Johnson et al., 2007; Mochihashi et al., 2009). The SIR algorithm, however, allows us to use an arbitrary proposal distribution Q for the samples. All that is needed is that we can calculate the true probability of each sample according to P, which is easily done using eq. 6. Our proposal distribution Q ignores the dependencies between the words within an utterance, as in eq. 7. This can be thought of as 'freezing' the model in order to determine a segmentation, just as in the PCFG proposal distribution of Johnson et al. (2007). Thus, the proposal segmentations and other quantities required to calculate the weights and propagate the particles are efficiently computable using the formulae in Figure 1 and the efficient algorithms described in detail in Johnson et al. (2007) and Mochihashi et al. (2009). Interestingly, even though the proposal distribution is only an approximation to the true posterior, Doucet et. al point out that as the number of particles goes to infinity, the approximation still converges to the target (Doucet et al., 2000).

Resampling A well known problem with Particle Filters is that after a number of observations most particles will be assigned very low weights which means that they contribute virtually nothing to the approximation of the target distribution. This is directly addressed by resampling steps in the SIR algorithm: whenever a quantity known as Effective Sample Size (ESS), approximated by eq. 12, falls below a certain threshold, say, $\frac{N}{2}$, the current set of particles is resampled by sampling with replacement from the current distribution over particles defined by the current weights. This results in high weight particles having multiple 'descendants', and in low weight particles being 'weeded out'. We experiment with two thresholds, N and $\frac{N}{2}$.

```
create N empty models $s_0^{(1)}$ to $s_0^{(N)}$
set initial weights $w_0^{(l)}$ to $\frac{1}{N}$
for example $i = 1 \to K$ do
    for particle $l = 1 \to N$ do
        sample $\sigma_i^{(l)} \sim Q_\sigma(\cdot \mid s_{i-1}^{(l)}, o_i, \alpha)$
        $s_i^{(l)} = s_{i-1}^{(l)} \bigcup \sigma_i^{(l)}$
        calculate the unnormalized particle weight $w_i^{*(l)}$
    end for
    calculate the normalized particle weights $w_i^{(l)}$ and
    calculate $\widehat{ESS}$
    if $\widehat{ESS} \leq THRESHOLD$ then
        resample all particles according to $w_i^{(l)}$
        set all weights to $\frac{1}{N}$
    end if
end for
```

Figure 2: Our Particle Filter algorithm. N is the number of particles, K is the number of examples. The formulae needed are given in Figure 1.

6 Experiments

We evaluate our algorithm along the lines of experiments discussed in Pearl et al. (2011), using Brent's version of the Bernstein corpus (Brent, 1999). Unlike them, however, we see no reason for actually splitting the data into training and test sets, following in this respect previous work such as Goldwater et al. (2009) by training and evaluating on the full corpus.

6.1 Evaluation

Evaluation is done for each learner by 'freezing' the model state after the learner has processed all examples and then sampling a proposed segmentation for each example o from $Q_\sigma(\cdot \mid s, o, \alpha)$, where s is the final model state. As we have multiple weighted final model states, the scores are calculated by evaluating each model state and then taking a weighted sum of the individual scores. This corresponds to taking an expectation over the posterior. Note that under the assumption that at the end of learning the 'lexicon' no longer changes, sampling from Q no longer constitutes an approximation as the intra-utterance dependencies Q ignores correspond to making changes to the lexicon.

As is common, we calculate precision, recall and the harmonic mean of the two, f-measure, for tokens, boundaries and elements in the lexicon. To illustrate these scores, assume the learner segmented the two utterances "the old woman" and "the old man" into "theold wo man" and "theold man". It has correctly identified 1 of the 6 tokens and predicted a total of 5 tokens, yielding a token recall of 1/6 and a token precision of 1/5. It has also identified 2 of the 4 boundaries, and has predicted a total of 3 boundaries, yielding a precision of 2/3 and a recall of 1/2. Lastly, it has correctly found only 1 of the 4 word types and predicted a total of 3 word types, giving a precision and a recall of of 1/3 and 1/4, respectively. So as to not clutter the table, we only report f-measure.

6.2 Experimental Results

There are two questions of interest with respect to the performance of the algorithm. First, we would like to know how faithful the algorithm is to the original model, i.e. whether the 'optimal' solution it finds is close to the 'optimal' solution according to the true model. For this, we compare the log-probability of the *training* data each algorithm assigns to it after learning. Second, we would like to know how well each algorithm performs in terms of the segmentation objectives.

For comparison to our Particle Filter algorithm, we used Pearl et al.'s (2011) implementation of their proposed online learners, the DPM and the DMCMC algorithm, and their implementation of a batch MCMC algorithm, a Metropolis Hastings sampler.[10] We set the threshold for resampling to N and $\frac{N}{2}$, ϕ to 0.02 and the remaining model parameters to the values reported in Goldwater et al. (2009): $\alpha = 20.0, \rho = 2.0$ for the unigram model, and $\alpha_0 = 100.0, \alpha_1 = 3000.0, p_\$ = 0.5$ for the bigram model.[11] In addition, we decided to not use annealing for either of the algorithms to get an idea of the 'raw' performance of their performance and simply run the unconstrained sampler for 20,000 iterations. While this does not reflect a 'true' ideal learner performance, it seems to us to be a reasonable method for an initial comparision, in particular since we have not yet applied ideas such as annealing or hyper-parameter sampling to the Particle Filter.

The particle filter results vary considerably, especially with small numbers of particles, which is why we report an average over multiple runs and report the standard deviation in brackets.[12] The results are given in Table 1.

6.3 Discussion

As could be expected, faithfulness to the original model increases with larger numbers of particles — the log-probability of the training-data seems to be positively related to N, though obviously non-linearly, and we expect that using even larger numbers of particles will bring the log-probability even closer to that assigned by the unconstrained learner. Also, the Particle Filters seem to work generally better for the Unigram model which is not very surprising, considering that "when tracking bigrams instead of just individual words, the learner's hypothesis space is much larger" (Pearl et al., 2011), and that Particle Filters are known to perform rather poorly in high dimensional state spaces.

In the Unigram setting, the Particle Filter is able to outperform the DPM algorithm in the 1000 particle / $\frac{N}{2}$ threshold setting, both in terms of log-probability and segmentation scores. With respect to the latter, it also outperforms the unconstrained learner in all but lexical f-measure. This is not very surprising, however, as Goldwater (2007) already found that sub-optimal solutions with respect to the actual model may be better with respect to segmentation, simply because the strong unigram assumption is so blatantly wrong.

For both the models, using a resampling threshold of $\frac{N}{2}$ instead of N seems to lead to better performance with respect to both measures, in particular in the Bigram setting. We are not sure how to interpret this result but have the suspicion that this difference may disappear as a larger number of particles is used and that what may be going on is that for 'small' numbers of particles, the diversity of samples drops too fast if resampling is applied after each observa-

[10]The scores reported in their paper apply only to their training-test split. We weren't able to run the DMCMC algorithm for the Unigram setting which is why we omit these scores — the scores reported for the Unigram learner on a five-fold training-test split in Pearl et al. are slightly better than for our best performing particle filter but not directly comparable, and no log-probability is given.

[11]$p_\$$ is the fixed utterance-boundary base-probability, ρ is the beta-prior on the utterance-boundary probability in the unigram model.

[12]For 1, 50 and 100 particles, we run 10 trials, for 500 and 1000 particles 2 trials. In principle, the unconstrained learners are also subject to some variance (Goldwater, 2007; Goldwater et al., 2009) but not to the extent of the particle filters.

	Learner		TF	BF	LF	log-prob$\times 10^3$
Unigram	MH-MCMC		63.11	80.29	**59.68**	**-209.57**
	DPM		65.65	80.05	44.96	-234.11
	PF	1	56.84 (4.36)	74.94 (2.92)	35.34 (3.21)	-244.28 (5.56)
		50	60.33/59.84 (5.82)/(6.00)	77.08/76.98 (3.68)/(3.75)	41.61/41.62 (3.40)/(3.15)	-240.38/-240.58 (6.43)/(5.18)
		100	62.97/61.02 (3.21)/(4.94)	79.05/77.87 (2.09)/(3.14)	43.04/43.61 (2.29)/(2.90)	-238.73/-238.92 (4.64)/(5.69)
		500	60.81/68.46 (7.05)/(1.87)	77.50/**82.27** (4.36)/(1.11)	45.25/47.52 (3.67)/(0.23)	-236.55/-231.85 (6.13)/(2.25)
		1000	64.11/**66.54** (4.84)/(5.31)	79.70/80.99 (2.74)/(3.24)	45.82/47.08 (1.36)/(2.90)	-234.87/**-231.93** (3.10)/(3.55)
Bigram	MH-MCMC		63.71	79.52	47.45	**-240.79**
	DMCMC		**70.78**	**83.97**	47.85	**-244.85**
	DPM		66.92	81.07	**52.54**	-250.52
	PF	1	48.55 (3.04)	71.22 (1.82)	35.02 (2.14)	-266.90 (2.40)
		50	55.98/57.85 (2.29)/(3.85)	75.21/76.49 (1.42)/(1.93)	43.34/45.21 (1.76)/(1.79)	-258.42/-256.16 (2.22)/(4.24)
		100	57.77/61.55 (2.77)/(2.06)	76.40/78.47 (1.45)/(1.30)	43.77/45.79 (1.46)/(1.50)	-257.93/-254.66 (2.03)/(1.47)
		500	57.99/63.58 (0.59)/(1.73)	76.33/80.05 (0.33)/(0.94)	44.70/47.82 (0.16)/(0.82)	-256.44/-252.14 (1.01)/(0.46)
		1000	57.88/61.76 (0.48)/(1.11)	76.55/78.30 (0.05)/(1.31)	46.93/49.33 (0.41)/(0.88)	-254.17/-251.33 (0.92)/(0.03)

Table 1: F-measure and log-probabilities on the Bernstein corpus for Pearl et al.'s (2011) batch MCMC algorithm (MH-MCMC), the online algorithms Dynamic Programming Maximization (DPM) and Delayed Markov Chain Monte Carlo (DMCMC), and our online Particle Filter (PF) with different numbers of particles (1, 50, 100, 500, 1000). TF, BF and LF stand for token, boundary and lexical f-measure, respectively. For the Particle Filter, numbers left of a '/' report scores for a resampling threshold of N, those on the right for $\frac{N}{2}$. Numbers in brackets are standard-deviations across multiple runs. Numbers in bold indicate the best overall performance, and the best performance of an online learner (if different from the best overall performance).

tion.

In the Bigram setting, even 1000 particles and a threshold of $\frac{N}{2}$ cannot outperform the conceptually much simpler DPM algorithm, let alone the DMCMC algorithm that comes pretty close to the unconstrained learner. The increased dimensionality of the state-space may require the use of even more particles, a point we plan to address in the future by optimizing our implementation so as to handle very large numbers of particles.

Also, there is no clear relation between the number of particles that are used and the variance in the results, in particular in the Unigram setting. While we are not sure about how to interpret this result, again it may have to do with the increased dimensionality: a possible explanation is that in the Unigram setting, 500 and 1000 particles already allow the model to explore very different hypotheses, leading to a larger variance in results, whereas in the Bigram setting, this number of particles only suffices to find solutions very close to each other.

All in all, however, the results suggest that using more particles gradually increases the learner's performance. This is unconditionally true for both settings in our experiments with respect to the log-probability; while this trend is not consistent for all segmentation scores, this simply reflects that the relation between log-probability and segmentation performance is not transparent, even for the Bigram model, as is clearly seen by the difference between the DMCMC and the Unconstrained learner.

Finally, we'd like to point out again that the DMCMC learner is, strictly speaking, not an online learner. Its ability to 'resample the past' corresponds closely to he idea of rejuvenation (Canini et al., 2009) in which each individual particle reconsiders past examples for a fixed amount of time at certain intervals and can, in principle, be added to our algorithm, something we plan to do in the future. Also, while the performance of the DPM algorithm for the Bigram model is rather impressive, it should be noted that the DPM algorithm embodies a heuristic greedy strategy that may or may not work in certain cases. While it obviously works rather well in this case, there is no mathematical or conceptual motivation for it and we can't be sure that its performance does not depend on accidental properties of the data.

7 Conclusion and Outlook

We have presented a Particle Filter algorithm for the Bayesian word segmentation models presented in Goldwater et al. (2009). The algorithm performs competitively with other proposed online algorithms for this kind of model, and as predicted, its performance increases with larger numbers of particles.

To our knowledge, it constitutes the first Particle Filter for this kind of model. Our formulation of the Particle Filter should extend to similarly complex Bayesian models in Computational Linguistics, e.g. the grammar models proposed in Johnson et al. (2007) and Liang et al. (2010), and may serve as a starting point for applying other Particle Filter algorithms to these models, a point we want to address in future research.

Also, while the strict online nature is desirable from a cognitive point of view, for practical purposes variants of Particle Filters that violate these strong assumptions, e.g. using the idea of rejuvenation that has previously been applied to Particle Filters for Latent Dirichlet Allocation in Canini et al. (2009), might offer considerable performance gains for practical NLP tasks, and we plan to extend our algorithm in this direction as well.

8 Acknowledgments

We would like to thank the reviewers for their helpful comments, and Lisa Pearl and Sharon Goldwater for sharing the source code for their learners and their data with us. This work was supported under the Australian Research Councils Discovery Projects funding scheme (project number DP110102506).

References

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

Michael R. Brent. 1999. An efficient, probabilistically sound algorithm for segmentation andword discovery. *Machine Learning - Special issue on natural language learning*, 34:71 – 105.

Kevin Canini, Lei Shi, and Thomas Griffiths. 2009. Online inference of topics with latent dirichlet allocation. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*.

Arnaud Doucet, Simon Godsill, and Christophe Andrieu. 2000. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10:197–208.

Sharon Goldwater, Thomas Griffiths, and Mark Johnson. 2009. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112:21–54.

Sharon Goldwater. 2007. *Nonparametric Bayesian Models of Lexical Acquisition*. Ph.D. thesis, Brown University.

Mark Johnson and Sharon Goldwater. 2009. Improving nonparameteric bayesian inference: experiments on unsupervised word segmentation with adaptor grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 317–325.

Mark Johnson, Thomas Griffiths, and Sharon Goldwater. 2007. Bayesian inference for pcfgs via markov chain monte carlo. In *Proceedings of NAACL HLT 2007*, pages 139–146.

Percy Liang and Dan Klein. 2009. Online em for unsupervised models. In *Proceedings of NAACL 2009*.

P. Liang, M. I. Jordan, and D. Klein. 2010. Probabilistic grammars and hierarchical dirichlet processes. In T. O'Hagan and M. West, editors, *The Handbook of Applied Bayesian Analysis*. Oxford University Press.

Bhaskara Marthi, Hanna Pasula, Stuart Russell, and Yuval Peres. 2002. Decayed mcmc filtering. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2002 (UAI-02)*.

Daichi Mochihashi, Takeshi Yamada, and Naonori Ueda. 2009. Bayesian unsupervised word segmentation with nested pitman-yor language modeling. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, page 100108.

Lisa Pearl, Sharon Goldwater, and Mark Steyvers. 2011. Online learning mechanisms for bayesian models of word segmentation. *Research on Language and Computation*, Special issue on computational models of language acquisition.

Anand Venkataraman. 2001. A statistical model for word discovery in transcribed speech. *Computational Linguistics*, 27:351–372.