Type-Based MCMC

Percy Liang
UC Berkeley
pliang@cs.berkeley.edu

Michael I. Jordan
UC Berkeley
jordan@cs.berkeley.edu

Dan Klein
UC Berkeley
klein@cs.berkeley.edu

Abstract

Most existing algorithms for learning latent-variable models—such as EM and existing Gibbs samplers—are *token-based*, meaning that they update the variables associated with one sentence at a time. The incremental nature of these methods makes them susceptible to local optima/slow mixing. In this paper, we introduce a *type-based sampler*, which updates a block of variables, identified by a *type*, which spans multiple sentences. We show improvements on part-of-speech induction, word segmentation, and learning tree-substitution grammars.

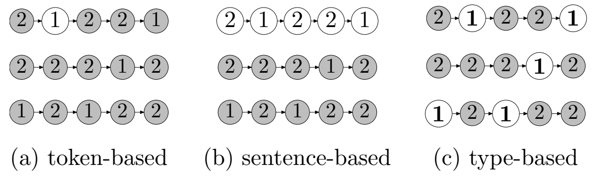


Figure 1: Consider a dataset of 3 sentences, each of length 5. Each variable is labeled with a *type* (1 or 2). The unshaded variables are the ones that are updated jointly by a sampler. The token-based sampler updates the variable for one token at a time (a). The sentence-based sampler updates all variables in a sentence, thus dealing with intra-sentential dependencies (b). The type-based sampler updates all variables of a particular type (1 in this example), thus dealing with dependencies due to common parameters (c).

1 Introduction

A long-standing challenge in NLP is the unsupervised induction of linguistic structures, for example, grammars from raw sentences or lexicons from phoneme sequences. A fundamental property of these unsupervised learning problems is multimodality. In grammar induction, for example, we could analyze subject-verb-object sequences as either *((subject verb) object)* (mode 1) or *(subject (verb object))* (mode 2).

Multimodality causes problems for token-based procedures that update variables for one example at a time. In EM, for example, if the parameters already assign high probability to the *((subject verb) object)* analysis, re-analyzing the sentences in E-step only reinforces the analysis, resulting in EM getting stuck in a local optimum. In (collapsed) Gibbs sampling, if all sentences are already analyzed as *((subject verb) object)*, sampling a sentence *conditioned* on all others will most likely not change its analysis, resulting in slow mixing.

To combat the problems associated with token-based algorithms, we propose a new sampling algorithm that operates on *types*. Our sampler would, for example, be able to change all occurrences of *((subject verb) object)* to *(subject (verb object))* in one step. These type-based operations are reminiscent of the type-based grammar operations of early chunk-merge systems (Wolff, 1988; Stolcke and Omohundro, 1994), but we work within a sampling framework for increased robustness.

In NLP, perhaps the the most simple and popular sampler is the *token-based Gibbs sampler*,[1] used in Goldwater et al. (2006), Goldwater and Griffiths (2007), and many others. By sampling only one

[1]In NLP, this is sometimes referred to as simply the collapsed Gibbs sampler.

variable at a time, this sampler is prone to slow mixing due to the strong coupling between variables. A general remedy is to sample blocks of coupled variables. For example, the *sentence-based sampler* samples all the variables associated with a sentence at once (e.g., the entire tag sequence). However, this blocking does not deal with the strong type-based coupling (e.g., all instances of a word should be tagged similarly). The type-based sampler we will present is designed exactly to tackle this coupling, which we argue is stronger and more important to deal with in unsupervised learning. Figure 1 depicts the updates made by each of the three samplers.

We tested our sampler on three models: a Bayesian HMM for part-of-speech induction (Goldwater and Griffiths, 2007), a nonparametric Bayesian model for word segmentation (Goldwater et al., 2006), and a nonparametric Bayesian model of tree substitution grammars (Cohn et al., 2009; Post and Gildea, 2009). Empirically, we find that type-based sampling improves performance and is less sensitive to initialization (Section 5).

2 Basic Idea via a Motivating Example

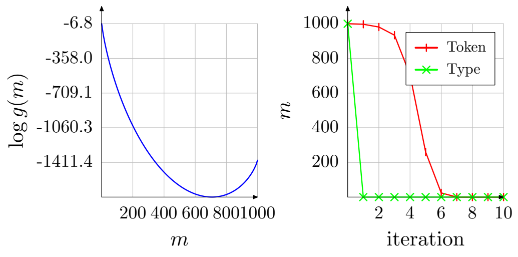
The key technical problem we solve in this paper is finding a block of variables which are both *highly coupled* and yet *tractable to sample jointly*. This section illustrates the main idea behind type-based sampling on a small word segmentation example.

Suppose our dataset $\mathbf{x}$ consists of n occurrences of the sequence *a b*. Our goal is infer $\mathbf{z} = (z_1, \dots, z_n)$, where $z_i = 0$ if the sequence is one word *ab*, and $z_i = 1$ if the sequence is two, *a* and *b*. We can model this situation with a simple generative model: for each $i = 1, \dots, n$, generate one or two words with equal probability. Each word is drawn independently based on probabilities $\theta = (\theta_a, \theta_b, \theta_{ab})$ which we endow with a uniform prior $\theta \sim \text{Dirichlet}(1, 1, 1)$.

We marginalize out θ to get the following standard expression (Goldwater et al., 2009):

$$p(\mathbf{z} \mid \mathbf{x}) \propto \frac{1^{(m)} 1^{(m)} 1^{(n-m)}}{3^{(n+m)}} \overset{\text{def}}{=} g(m), \qquad (1)$$

where $m = \sum_{i=1}^{n} z_i$ is the number of two-word sequences and $a^{(k)} = a(a+1) \cdots (a+k-1)$ is the *ascending factorial*.[2] Figure 2(a) depicts the resulting bimodal posterior.



(a) bimodal posterior (b) sampling run

Figure 2: (a) The posterior (1) is sharply bimodal (note the log-scale). (b) A run of the token-based and type-based samplers. We initialize both samplers with $m = n$ ($n = 1000$). The type-based sampler mixes instantly (in fact, it makes independent draws from the posterior) whereas the token-based sampler requires five passes through the data before finding the high probability region $m \approxeq 0$.

A token-based sampler chooses one z_i to update according to the posterior $p(z_i \mid \mathbf{z}_{-i}, \mathbf{x})$. To illustrate the mixing problem, consider the case where $m = n$, i.e., all sequences are analyzed as two words. From (1), we can verify that $p(z_i = 0 \mid \mathbf{z}_{-i}, \mathbf{x}) = O(\frac{1}{n})$. When $n = 1000$, this means that there is only a 0.002 probability of setting $z_i = 0$, a very unlikely but necessary first step to take to escape this local optimum. Indeed, Figure 2(b) shows how the token-based sampler requires five passes over the data to finally escape.

Type-based sampling completely eradicates the local optimum problem in this example. Let us take a closer look at (1). Note that $p(\mathbf{z} \mid \mathbf{x})$ only depends on a single integer m, which only takes one of $n+1$ values, not on the particular $\mathbf{z}$. This shows that the z_is are *exchangeable*. There are $\binom{n}{m}$ possible values of $\mathbf{z}$ satisfying $m = \sum_i z_i$, each with *the same* probability $g(m)$. Summing, we get:

$$p(m \mid \mathbf{x}) \propto \sum_{\mathbf{z}: m = \sum_i z_i} p(\mathbf{x}, \mathbf{z}) = \binom{n}{m} g(m). \qquad (2)$$

A sampling strategy falls out naturally: First, sample the number m via (2). Conditioned on m, choose

[2]The ascending factorial function arises from marginalizing Dirichlet distributions and is responsible the rich-gets-richer phenomenon: the larger n is, more we gain by increasing it.

the particular $\mathbf{z}$ uniformly out of the $\binom{n}{m}$ possibilities. Figure 2(b) shows the effectiveness of this type-based sampler.

This simple example exposes the fundamental challenge of multimodality in unsupervised learning. Both $m = 0$ and $m = n$ are modes due to the rich-gets-richer property which arises by virtue of all n examples sharing the same parameters $\boldsymbol{\theta}$. This sharing is a double-edged sword: It provides us with clustering structure but also makes inference hard. Even though $m = n$ is much worse (by a factor exponential in n) than $m = 0$, a naïve algorithm can easily have trouble escaping $m = n$.

3 Setup

We will now present the type-based sampler in full generality. Our sampler is applicable to any model which is built out of local multinomial choices, where each multinomial has a Dirichlet process prior (a Dirichlet prior if the number of choices is finite). This includes most probabilistic models in NLP (excluding ones built from log-linear features).

As we develop the sampler, we will provide concrete examples for the Bayesian hidden Markov model (HMM), the Dirichlet process unigram segmentation model (USM) (Goldwater et al., 2006), and the probabilistic tree-substitution grammar (PTSG) (Cohn et al., 2009; Post and Gildea, 2009).

3.1 Model parameters

A model is specified by a collection of multinomial parameters $\boldsymbol{\theta} = \{\theta_r\}_{r\in\mathcal{R}}$, where $\mathcal{R}$ is an index set. Each vector θ_r specifies a distribution over outcomes: outcome o has probability θ_{ro}.

- HMM: Let K is the number of states. The set $\mathcal{R} = \{(q,k) : q \in \{\mathrm{T},\mathrm{E}\}, k = 1,\dots,K\}$ indexes the K transition distributions $\{\theta_{(\mathrm{T},k)}\}$ (each over outcomes $\{1,\dots,K\}$) and K emission distributions $\{\theta_{(\mathrm{E},k)}\}$ (each over the set of words).
- USM: $\mathcal{R} = \{0\}$, and θ_0 is a distribution over (an infinite number of) words.
- PTSG: $\mathcal{R}$ is the set of grammar symbols, and each θ_r is a distribution over labeled tree fragments with root label r.

$\mathcal{R}$	index set for parameters
$\boldsymbol{\theta} = \{\theta_r\}_{r\in\mathcal{R}}$	multinomial parameters
$\boldsymbol{\mu} = \{\mu_r\}_{r\in\mathcal{R}}$	base distributions (fixed)
$\mathcal{S}$	set of sites
$\mathbf{b} = \{b_s\}_{s\in\mathcal{S}}$	binary variables (to be sampled)
$\mathbf{z}$	latent structure (set of choices)
$\mathbf{z}^{-s}$	choices not depending on site s
$\mathbf{z}^{s:b}$	choices after setting $b_s = b$
$\Delta\mathbf{z}^{s:b}$	$\mathbf{z}^{s:b}\backslash\mathbf{z}^{-s}$: new choices from $b_s = b$
$S \subset \mathcal{S}$	sites selected for sampling
m	# sites in S assigned $b_s = 1$
$\mathbf{n} = \{n_{ro}\}$	counts (sufficient statistics of $\mathbf{z}$)

Table 1: Notation used in this paper. Note that there is a one-to-one mapping between $\mathbf{z}$ and $(\mathbf{b},\mathbf{x})$. The information relevant for evaluating the likelihood is $\mathbf{n}$. We use the following parallel notation: $\mathbf{n}^{-s} = \mathbf{n}(\mathbf{z}^{-s}), \mathbf{n}^{s:b} = \mathbf{n}(\mathbf{z}^{s:b}), \Delta\mathbf{n}^{s} = \mathbf{n}(\Delta\mathbf{z}^{s})$.

3.2 Choice representation of latent structure z

We represent the latent structure $\mathbf{z}$ as a set of local *choices*:[3]

- HMM: $\mathbf{z}$ contains elements of the form (T, i, a, b), denoting a transition from state a at position i to state b at position $i + 1$; and (E, i, a, w), denoting an emission of word w from state a at position i.
- USM: $\mathbf{z}$ contains elements of the form (i, w), denoting the generation of word w at character position i extending to position $i + |w| - 1$.
- PTSG: $\mathbf{z}$ contains elements of the form (x, t), denoting the generation of tree fragment t rooted at node x.

The choices $\mathbf{z}$ are connected to the parameters $\boldsymbol{\theta}$ as follows: $p(\mathbf{z} \mid \boldsymbol{\theta}) = \prod_{z\in\mathbf{z}} \theta_{z.r,z.o}$. Each choice $z \in \mathbf{z}$ is identified with some $z.r \in \mathcal{R}$ and outcome $z.o$. Intuitively, choice z was made by drawing drawing $z.o$ from the multinomial distribution $\theta_{z.r}$.

3.3 Prior

We place a Dirichlet process prior on θ_r (Dirichlet prior for finite outcome spaces): $\theta_r \sim \mathrm{DP}(\alpha_r, \mu_r)$, where α_r is a concentration parameter and μ_r is a fixed base distribution.

[3]We assume that $\mathbf{z}$ contains both a latent part and the observed input $\mathbf{x}$, i.e., $\mathbf{x}$ is a deterministic function of $\mathbf{z}$.

Let $n_{ro}(\mathbf{z}) = |\{z \in \mathbf{z} : z.r = r, z.o = o\}|$ be the number of draws from θ_r resulting in outcome o, and $n_{r\cdot} = \sum_o n_{ro}$ be the number of times θ_r was drawn from. Let $\mathbf{n}(\mathbf{z}) = \{n_{ro}(\mathbf{z})\}$ denote the vector of sufficient statistics associated with choices $\mathbf{z}$. When it is clear from context, we simply write $\mathbf{n}$ for $\mathbf{n}(\mathbf{z})$. Using these sufficient statistics, we can write $p(\mathbf{z} \mid \boldsymbol{\theta}) = \prod_{r,o} \theta_{ro}^{n_{ro}(\mathbf{z})}$.

We now marginalize out $\boldsymbol{\theta}$ using Dirichlet-multinomial conjugacy, producing the following expression for the likelihood:

$$p(\mathbf{z}) = \prod_{r \in \mathcal{R}} \frac{\prod_o (\alpha_{ro}\mu_{ro})^{(n_{ro}(\mathbf{z}))}}{\alpha_r{}^{(n_{r\cdot}(\mathbf{z}))}}, \tag{3}$$

where $a^{(k)} = a(a+1)\cdots(a+k-1)$ is the ascending factorial. (3) is the distribution that we will use for sampling.

4 Type-Based Sampling

Having described the setup of the model, we now turn to posterior inference of $p(\mathbf{z} \mid \mathbf{x})$.

4.1 Binary Representation

We first define a new representation of the latent structure based on binary variables $\mathbf{b}$ so that there is a bijection between $\mathbf{z}$ and $(\mathbf{b}, \mathbf{x})$; $\mathbf{z}$ was used to define the model, $\mathbf{b}$ will be used for inference. We will use $\mathbf{b}$ to exploit the ideas from Section 2. Specifically, let $\mathbf{b} = \{b_s\}_{s \in \mathcal{S}}$ be a collection of binary variables indexed by a set of *sites* $\mathcal{S}$.

- HMM: If the HMM has $K = 2$ states, $\mathcal{S}$ is the set of positions in the sequence. For each $s \in \mathcal{S}$, b_s is the hidden state at s. The extension to general K is considered at the end of Section 4.4.
- USM: $\mathcal{S}$ is the set of non-final positions in the sequence. For each $s \in \mathcal{S}$, b_s denotes whether a word boundary exists between positions s and $s + 1$.
- PTSG: $\mathcal{S}$ is the set of internal nodes in the parse tree. For $s \in \mathcal{S}$, b_s denotes whether a tree fragment is rooted at node s.

For each site $s \in \mathcal{S}$, let $\mathbf{z}^{s:0}$ and $\mathbf{z}^{s:1}$ denote the choices associated with the structures obtained by setting the binary variable $b_s = 0$ and $b_s = 1$, respectively. Define $\mathbf{z}^{-s} \overset{\text{def}}{=} \mathbf{z}^{s:0} \cap \mathbf{z}^{s:1}$ to be the set of choices that do not *depend* on the value of b_s, and $\mathbf{n}^{-s} \overset{\text{def}}{=} \mathbf{n}(\mathbf{z}^{-s})$ be the corresponding counts.

- HMM: $\mathbf{z}^{-s}$ includes all *but* the transitions into and out of the state at s plus the emission at s.
- USM: $\mathbf{z}^{-s}$ includes all except the word ending at s and the one starting at $s + 1$ if there is a boundary ($b_s = 1$); except the word covering s if no boundary exists ($b_s = 0$).
- PTSG: $\mathbf{z}^{-s}$ includes all except the tree fragment rooted at node s and the one with leaf s if $b_s = 1$; except the single fragment containing s if $b_s = 0$.

4.2 Sampling One Site

A token-based sampler considers one site s at a time. Specifically, we evaluate the likelihoods of $\mathbf{z}^{s:0}$ and $\mathbf{z}^{s:1}$ according to (3) and sample b_s with probability proportional to the likelihoods. Intuitively, this can be accomplished by removing choices that depend on b_s (resulting in $\mathbf{z}^{-s}$), evaluating the likelihood resulting from setting b_s to 0 or 1, and then adding the appropriate choices back in.

More formally, let $\Delta\mathbf{z}^{s:b} \overset{\text{def}}{=} \mathbf{z}^{s:b} \backslash \mathbf{z}^{-s}$ be the new choices that would be added if we set $b_s = b \in \{0, 1\}$, and let $\Delta\mathbf{n}^{s:b} \overset{\text{def}}{=} \mathbf{n}(\Delta\mathbf{z}^{s:b})$ be the corresponding counts. With this notation, we can write the posterior as follows:

$$p(b_s = b \mid \mathbf{b} \backslash \mathbf{b}_s) \propto \prod_{r \in \mathcal{R}} \frac{\prod_o (\alpha_{ro}\mu_{ro} + n_{ro}^{-s})^{(\Delta n_{ro}^{s:b})}}{(\alpha_r + n_{r\cdot}^{-s})^{(\Delta n_{r\cdot}^{s:b})}}. \tag{4}$$

The form of the conditional (4) follows from the joint (3) via two properties: additivity of counts ($\mathbf{n}^{s:b} = \mathbf{n}^{-s} + \Delta\mathbf{n}^{s:b}$) and a simple property of ascending factorials ($a^{(k+\delta)} = a^{(k)}(a + k)^{(\delta)}$).

In practice, most of the entries of $\Delta\mathbf{n}^{s:b}$ are zero. For the HMM, $n_{ro}^{s:b}$ would be nonzero only for the transitions into the new state (b) at position s ($z_{s-1} \to b$), transitions out of that state ($b \to z_{s+1}$), and emissions from that state ($b \to x_s$).

4.3 Sampling Multiple Sites

We would like to sample multiple sites jointly as in Section 2, but we cannot choose any arbitrary subset $S \subset \mathcal{S}$, as the likelihood will in general depend on the exact assignment of $\mathbf{b}_S \overset{\text{def}}{=} \{b_s\}_{s \in S}$, of which

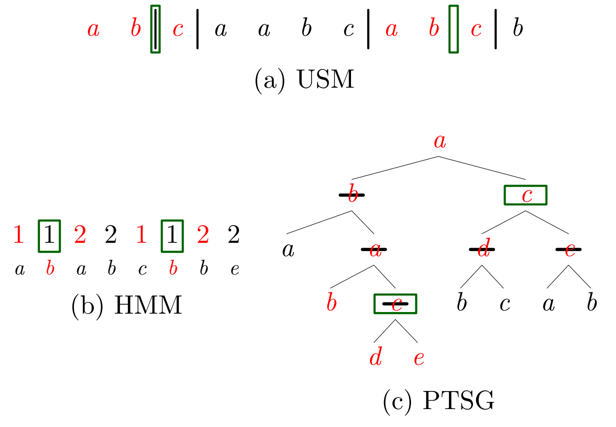


Figure 3: The type-based sampler jointly samples all variables at a set of sites S (in green boxes). Sites in S are chosen based on types (denoted in red). (a) HMM: two sites have the same type if they have the same previous and next states and emit the same word; they conflict unless separated by at least one position. (b) USM: two sites have the same type if they are both of the form $ab|c$ or abc; note that occurrences of the same letters with other segmentations do not match the type. (c) PTSG: analogous to the USM, only for tree rather than sequences.

there are an exponential number. To exploit the exchangeability property in Section 2, we need to find sites which look "the same" from the model's point of view, that is, the likelihood only depends on $\mathbf{b}_S$ via $m \overset{\text{def}}{=} \sum_{s \in S} b_s$.

To do this, we need to define two notions, *type* and *conflict*. We say sites s and s' have the same *type* if the counts added by setting either b_s or $b_{s'}$ are the same, that is, $\Delta\mathbf{n}^{s:b} = \Delta\mathbf{n}^{s':b}$ for $b \in \{0, 1\}$. This motivates the following definition of the *type* of site s with respect to $\mathbf{z}$:

$$t(\mathbf{z}, s) \overset{\text{def}}{=} (\Delta\mathbf{n}^{s:0}, \Delta\mathbf{n}^{s:1}), \tag{5}$$

We say that s and s' have the same type if $t(\mathbf{z}, s) = t(\mathbf{z}, s')$. Note that the actual choices added ($\Delta\mathbf{z}^{s:b}$ and $\Delta\mathbf{z}^{s':b}$) are in general different as s and s' correspond to different parts of the latent structure, but the model only depends on counts and is indifferent to this. Figure 3 shows examples of same-type sites for our three models.

However, even if all sites in S have the same type, we still cannot sample $\mathbf{b}_S$ jointly, since changing one b_s might change the type of another site s'; indeed, this dependence is reflected in (5), which shows that types depend on $\mathbf{z}$. For example, $s, s' \in S$ conflict when $s' = s + 1$ in the HMM or when s and s' are boundaries of one segment (USM) or one tree fragment (PTSG). Therefore, one additional concept is necessary: We say two sites s and s' *conflict* if there is some choice that depends on both b_s and $b_{s'}$; formally, $(\mathbf{z} \backslash \mathbf{z}^{-s}) \cap (\mathbf{z} \backslash \mathbf{z}^{-s'}) \neq \emptyset$.

Our key mathematical result is as follows:

Proposition 1 *For any set $S \subset \mathcal{S}$ of non-conflicting sites with the same type,*

$$p(\mathbf{b}_S \mid \mathbf{b} \backslash \mathbf{b}_S) \propto g(m) \tag{6}$$

$$p(m \mid \mathbf{b} \backslash \mathbf{b}_S) \propto \binom{|S|}{m} g(m), \tag{7}$$

for some easily computable $g(m)$, where $m = \sum_{s \in S} b_s$.

We will derive $g(m)$ shortly, but first note from (6) that the likelihood for a particular setting of $\mathbf{b}_S$ depends on $\mathbf{b}_S$ only via m as desired. (7) sums over all $\binom{|S|}{m}$ settings of $\mathbf{b}_S$ with $m = \sum_{s \in S} b_s$. The algorithmic consequences of this result is that to sample $\mathbf{b}_S$, we can first compute (7) for each $m \in \{0, \dots, |S|\}$, sample m according to the normalized distribution, and then choose the actual $\mathbf{b}_S$ uniformly subject to m.

Let us now derive $g(m)$ by generalizing (4). Imagine removing all sites S and their dependent choices and adding in choices corresponding to some assignment $\mathbf{b}_S$. Since all sites in S are non-conflicting and of the same type, the count contribution $\Delta\mathbf{n}^{s:b}$ is the same for every $s \in S$ (i.e., sites in S are exchangeable). Therefore, the likelihood of the new assignment $\mathbf{b}_S$ depends only on the new counts:

$$\Delta\mathbf{n}^{S:m} \overset{\text{def}}{=} m\Delta\mathbf{n}^{s:1} + (|S| - m)\Delta\mathbf{n}^{s:0}. \tag{8}$$

Using these new counts in place of the ones in (4), we get the following expression:

$$g(m) = \prod_{r \in \mathcal{R}} \frac{\prod_o (\alpha_{ro}\mu_{ro} + n_{ro}(\mathbf{z}^{-S}))^{(\Delta n_{ro}^{S:m})}}{\alpha_r + n_{r\cdot}(\mathbf{z}^{-S})^{(\Delta n_{r\cdot}^{S:m})}}. \tag{9}$$

4.4 Full Algorithm

Thus far, we have shown how to sample $\mathbf{b}_S$ given a set $S \subset \mathcal{S}$ of non-conflicting sites with the same type. To complete the description of the type-based

```
Type-Based Sampler
for each iteration $t = 1, \dots, T$:
  for each pivot site $s_0 \in \mathcal{S}$:
    $S \leftarrow \text{TB}(\mathbf{z}, s_0)$ ($S$ is the type block centered at $s_0$)
    decrement $\mathbf{n}$ and remove from $\mathbf{z}$ based on $\mathbf{b}_S$
    sample $m$ according to (7)
    sample $M \subset S$ with $|M| = m$ uniformly at random
    set $b_s = \mathbb{I}[s \in M]$ for each $s \in S$
    increment $\mathbf{n}$ and add to $\mathbf{z}$ accordingly
```

Figure 4: Pseudocode for the general type-based sampler. We operate in the binary variable representation $\mathbf{b}$ of $\mathbf{z}$. Each step, we jointly sample $|S|$ variables (of the same type).

sampler, we need to specify how to choose S. Our general strategy is to first choose a *pivot site* $s_0 \in \mathcal{S}$ uniformly at random and then set $S = \text{TB}(\mathbf{z}, s_0)$ for some function TB. Call S the *type block* centered at s_0. The following two criteria on TB are sufficient for a valid sampler: (A) $s_0 \in S$, and (B) the type blocks are *stable*, which means that if we change $\mathbf{b}_S$ to any $\mathbf{b}'_S$ (resulting in a new $\mathbf{z}'$), the type block centered at s_0 with respect to $\mathbf{z}'$ does not change (that is, $\text{TB}(\mathbf{z}', s_0) = S$). (A) ensures ergodicity; (B), reversibility.

Now we define TB as follows: First set $S = \{s_0\}$. Next, loop through all sites $s \in \mathcal{S}$ with the same type as s_0 in some fixed order, adding s to S if it does not conflict with any sites already in S. Figure 4 provides the pseudocode for the full algorithm.

Formally, this sampler cycles over $|\mathcal{S}|$ transition kernels, one for each pivot site. Each kernel (indexed by $s_0 \in \mathcal{S}$) defines a blocked Gibbs move, i.e. sampling from $p(\mathbf{b}_{\text{TB}(\mathbf{z},s_0)} \mid \cdots)$.

Efficient Implementation There are two operations we must perform efficiently: (A) looping through sites with the same type as the pivot site s_0, and (B) checking whether such a site s conflicts with any site in S. We can perform (B) in $O(1)$ time by checking if any element of $\Delta\mathbf{z}^{s:b_s}$ has already been removed; if so, there is a conflict and we skip s. To do (A) efficiently, we maintain a hash table mapping type t to a doubly-linked list of sites with type t. There is an $O(1)$ cost for maintaining this data structure: When we add or remove a site s, we just need to add or remove neighboring sites s' from their respective linked lists, since their types depend on b_s. For example, in the HMM, when we remove site s, we also remove sites $s-1$ and $s+1$.

For the USM, we use a simpler solution: maintain a hash table mapping each word w to a list of positions where w occurs. Suppose site (position) s straddles words a and b. Then, to perform (A), we retrieve the list of positions where *a*, *b*, and *ab* occur, intersecting the a and b lists to obtain a list of positions where $a\ b$ occurs. While this intersection is often much smaller than the pre-intersected lists, we found in practice that the smaller amount of bookkeeping balanced out the extra time spent intersecting. We used a similar strategy for the PTSG, which significantly reduces the amount of bookkeeping.

Skip Approximation Large type blocks mean larger moves. However, such a block S is also sampled more frequently—once for every choice of a pivot site $s_0 \in S$. However, we found that empirically, $\mathbf{b}_S$ changes very infrequently. To eliminate this apparent waste, we use the following approximation of our sampler: do not consider $s_0 \in \mathcal{S}$ as a pivot site if s_0 belongs to some block which was already sampled in the current iteration. This way, each site is considered roughly once per iteration.[4]

Sampling Non-Binary Representations We can sample in models without a natural binary representation (e.g., HMMs with with more than two states) by considering random binary slices. Specifically, suppose $b_s \in \{1, \dots, K\}$ for each site $s \in \mathcal{S}$. We modify Figure 4 as follows: After choosing a pivot site $s_0 \in \mathcal{S}$, let $k = b_{s_0}$ and choose k' uniformly from $\{1, \dots, K\}$. Only include sites in one of these two states by re-defining the type block to be $S = \{s \in \text{TB}(\mathbf{z}, s_0) : b_s \in \{k, k'\}\}$, and sample $\mathbf{b}_S$ restricted to these two states by drawing from $p(\mathbf{b}_S \mid \mathbf{b}_S \in \{k, k'\}^{|S|}, \cdots)$. By choosing a random k' each time, we allow $\mathbf{b}$ to reach any point in the space, thus achieving ergodicity just by using these binary restrictions.

5 Experiments

We now compare our proposed type-based sampler to various alternatives, evaluating on marginal like-

[4]A site could be sampled more than once if it belonged to more than one type block during the iteration (recall that types depend on $\mathbf{z}$ and thus could change during sampling).

lihood (3) and accuracy for our three models:

- HMM: We learned a $K = 45$ state HMM on the Wall Street Journal (WSJ) portion of the Penn Treebank (49208 sentences, 45 tags) for part-of-speech induction. We fixed α_r to 0.1 and μ_r to uniform for all r.

  For accuracy, we used the standard metric based on greedy mapping, where each state is mapped to the POS tag that maximizes the number of correct matches (Haghighi and Klein, 2006). We did not use a tagging dictionary.

- USM: We learned a USM model on the Bernstein-Ratner corpus from the CHILDES database used in Goldwater et al. (2006) (9790 sentences) for word segmentation. We fixed α_0 to 0.1. The base distribution μ_0 penalizes the length of words (see Goldwater et al. (2009) for details). For accuracy, we used word token F_1.

- PTSG: We learned a PTSG model on sections 2–21 of the WSJ treebank.[5] For accuracy, we used EVALB parsing F_1 on section 22.[6] Note this is a supervised task with latent-variables, whereas the other two are purely unsupervised.

5.1 Basic Comparison

Figure 5(a)–(c) compares the likelihood and accuracy (we use the term *accuracy* loosely to also include F_1). The initial observation is that the type-based sampler (TYPE) outperforms the token-based sampler (TOKEN) across all three models on both metrics.

We further evaluated the PTSG on parsing. Our standard treebank PCFG estimated using maximum likelihood obtained 79% F_1. TOKEN obtained an F_1 of 82.2%, and TYPE obtained a comparable F_1 of 83.2%. Running the PTSG for longer continued to improve the likelihood but actually hurt parsing accuracy, suggesting that the PTSG model is overfitting.

To better understand the gains from TYPE over TOKEN, we consider three other alternative samplers. First, annealing ($\text{TOKEN}_{\text{anneal}}$) is a commonly-used technique to improve mixing, where (3) is raised to some inverse temperature.[7] In Figure 5(a)–(c), we see that unlike TYPE, $\text{TOKEN}_{\text{anneal}}$ does not improve over TOKEN uniformly: it hurts for the HMM, improves slightly for the USM, and makes no difference for the PTSG. Although annealing does increase mobility of the sampler, this mobility is undirected, whereas type-based sampling increases mobility in purely model-driven directions.

Unlike past work that operated on types (Wolff, 1988; Brown et al., 1992; Stolcke and Omohundro, 1994), type-based sampling makes stochastic choices, and moreover, these choices are reversible. Is this stochasticity important? To answer this, we consider a variant of TYPE, $\text{TYPE}_{\text{greedy}}$: instead of sampling from (7), $\text{TYPE}_{\text{greedy}}$ considers a type block S and sets b_s to 0 for all $s \in S$ if $p(\mathbf{b}_S = (0, \dots, 0) \mid \cdots) > p(\mathbf{b}_S = (1, \dots, 1) \mid \cdots)$; else it sets b_s to 1 for all $s \in S$. From Figure 5(a)–(c), we see that greediness is disastrous for the HMM, hurts a little for USM, and makes no difference on the PTSG. These results show that stochasticity can indeed be important.

We consider another block sampler, SENTENCE, which uses dynamic programming to sample all variables in a sentence (using Metropolis-Hastings to correct for intra-sentential type-level coupling). For USM, we see that SENTENCE performs worse than TYPE and is comparable to TOKEN, suggesting that type-based dependencies are stronger and more important to deal with than intra-sentential dependencies.

5.2 Initialization

We initialized all samplers as follows: For the USM and PTSG, for each site s, we place a boundary (set $b_s = 1$) with probability η. For the HMM, we set b_s to state 1 with probability η and a random state with

[5]Following Petrov et al. (2006), we performed an initial preprocessing step on the trees involving Markovization, binarization, and collapsing of unary chains; words occurring once are replaced with one of 50 "unknown word" tokens, using base distributions $\{\mu_r\}$ that penalize the size of trees, and sampling the hyperparameters (see Cohn et al. (2009) for details).

[6]To evaluate, we created a grammar where the rule probabilities are the mean values under the PTSG distribution: this involves taking a weighted combination (based on the concentration parameters) of the rule counts from the PTSG samples and the PCFG-derived base distribution. We used the decoder of DeNero et al. (2009) to parse.

[7]We started with a temperature of 10 and gradually decreased it to 1 during the first half of the run, and kept it at 1 thereafter.

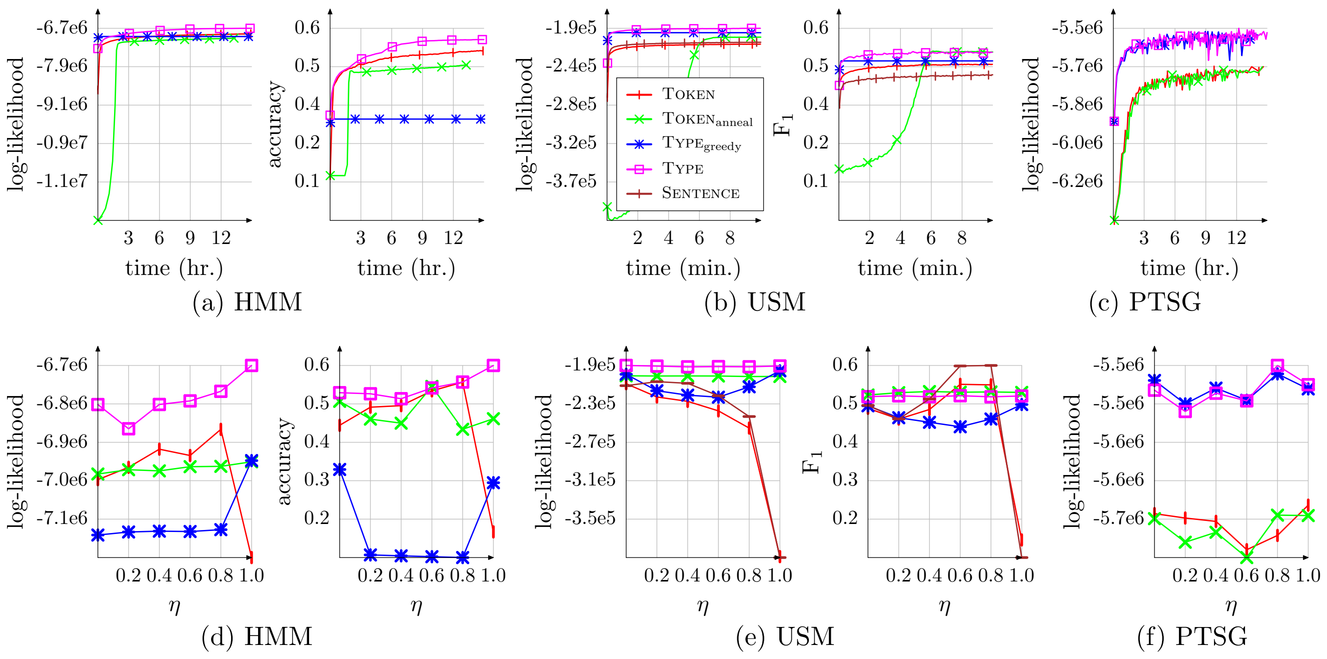


Figure 5: (a)–(c): Log-likelihood and accuracy over time. Type performs the best. Relative to Type, Type$_{\text{greedy}}$ tends to hurt performance. Token generally works worse. Relative to Token, Token$_{\text{anneal}}$ produces mixed results. Sentence behaves like Token. (d)–(f): Effect of initialization. The metrics were applied to the current sample after 15 hours for the HMM and PTSG and 10 minutes for the USM. Type generally prefers larger η and outperform the other samplers.

probability $1 - \eta$. Results in Figure 5(a)–(c) were obtained by setting η to maximize likelihood.

Since samplers tend to be sensitive to initialization, it is important to explore the effect of initialization (parametrized by $\eta \in [0, 1]$). Figure 5(d)–(f) shows that Type is consistently the best, whereas other samplers can underperform Type by a large margin. Note that Type favors $\eta = 1$ in general. This setting maximizes the number of initial types, and thus creates larger type blocks and thus enables larger moves. Larger type blocks also mean more dependencies that Token is unable to deal with.

6 Related Work and Discussion

Block sampling, on which our work is built, is a classical idea, but is used restrictively since sampling large blocks is computationally expensive. Past work for clustering models maintained tractability by using Metropolis-Hastings proposals (Dahl, 2003) or introducing auxiliary variables (Swendsen and Wang, 1987; Liang et al., 2007). In contrast, our type-based sampler simply identifies tractable blocks based on exchangeability.

Other methods for learning latent-variable models include EM, variational approximations, and uncollapsed samplers. All of these methods maintain distributions over (or settings of) the latent variables of the model and update the representation iteratively (see Gao and Johnson (2008) for an overview in the context of POS induction). However, these methods are at the core all token-based, since they only update variables in a single example at a time.[8]

Blocking variables by type—the key idea of this paper—is a fundamental departure from token-based methods. Though type-based changes have also been proposed (Brown et al., 1992; Stolcke and Omohundro, 1994), these methods operated greedily, and in Section 5.1, we saw that being greedy led to more brittle results. By working in a sampling framework, we were able bring type-based changes to fruition.

[8]While EM technically updates all distributions over latent variables in the E-step, this update is performed *conditioned on* model parameters; it is this coupling (made more explicit in collapsed samplers) that makes EM susceptible to local optima.

References

P. F. Brown, V. J. D. Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18:467–479.

T. Cohn, S. Goldwater, and P. Blunsom. 2009. Inducing compact but accurate tree-substitution grammars. In *North American Association for Computational Linguistics (NAACL)*, pages 548–556.

D. B. Dahl. 2003. An improved merge-split sampler for conjugate Dirichlet process mixture models. Technical report, Department of Statistics, University of Wisconsin.

J. DeNero, M. Bansal, A. Pauls, and D. Klein. 2009. Efficient parsing for transducer grammars. In *North American Association for Computational Linguistics (NAACL)*, pages 227–235.

J. Gao and M. Johnson. 2008. A comparison of Bayesian estimators for unsupervised hidden Markov model POS taggers. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 344–352.

S. Goldwater and T. Griffiths. 2007. A fully Bayesian approach to unsupervised part-of-speech tagging. In *Association for Computational Linguistics (ACL)*.

S. Goldwater, T. Griffiths, and M. Johnson. 2006. Contextual dependencies in unsupervised word segmentation. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*.

S. Goldwater, T. Griffiths, and M. Johnson. 2009. A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112:21–54.

A. Haghighi and D. Klein. 2006. Prototype-driven learning for sequence models. In *North American Association for Computational Linguistics (NAACL)*, pages 320–327.

P. Liang, M. I. Jordan, and B. Taskar. 2007. A permutation-augmented sampler for Dirichlet process mixture models. In *International Conference on Machine Learning (ICML)*.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*, pages 433–440.

M. Post and D. Gildea. 2009. Bayesian learning of a tree substitution grammar. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

A. Stolcke and S. Omohundro. 1994. Inducing probabilistic grammars by Bayesian model merging. In *International Colloquium on Grammatical Inference and Applications*, pages 106–118.

R. H. Swendsen and J. S. Wang. 1987. Nonuniversal critical dynamics in MC simulations. *Physics Review Letters*, 58:86–88.

J. G. Wolff. 1988. Learning syntax and meanings through optimization and distributional analysis. In *Categories and processes in language acquisition*, pages 179–215.