Grammar induction from (lots of) words alone

John K Pate
Department of Linguistics
The University at Buffalo
Buffalo, NY 14260, USA
johnpate@buffalo.edu

Mark Johnson
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
mark.johnson@mq.edu.au

Abstract

Grammar induction is the task of learning syntactic structure in a setting where that structure is hidden. Grammar induction from words alone is interesting because it is similiar to the problem that a child learning a language faces. Previous work has typically assumed richer but cognitively implausible input, such as POS tag annotated data, which makes that work less relevant to human language acquisition. We show that grammar induction from words alone is in fact feasible when the model is provided with sufficient training data, and present two new streaming or mini-batch algorithms for PCFG inference that can learn from millions of words of training data. We compare the performance of these algorithms to a batch algorithm that learns from less data. The minibatch algorithms outperform the batch algorithm, showing that cheap inference with more data is better than intensive inference with less data. Additionally, we show that the harmonic initialiser, which previous work identified as essential when learning from small POS-tag annotated corpora (Klein and Manning, 2004), is not superior to a uniform initialisation.

1 Introduction

How children acquire the syntax of the languages they ultimately speak is a deep scientific question of fundamental importance to linguistics and cognitive science (Chomsky, 1986). The natural language processing task of grammar induction in principle should provide models for how children do this. However, previous work on grammar induction has learned from small datasets, and has dealt with the resulting data sparsity by modifying the input and using careful search heuristics. While these techniques are useful from an engineering perspective, they make the models less relevant to human language acquisition.

In this paper, we use scalable algorithms for Probabilistic Context Free Grammar (PCFG) inference to perform grammar induction from millions of words of speech transcripts, and show that grammar induction from words alone is both feasible and insensitive to initialization. To ensure the robustness of our results, we use two algorithms for Variational Bayesian PCFG inference, and adapt two algorithms that have been proposed for Latent Dirichlet Allocation (LDA) topic models. Most importantly, we find that the three algorithms that scale to large datasets improve steadily over training to about the same predictive probability and parsing performance.

Moreover, while grammar induction from small datasets of POS-tagged newswire text fails without careful 'harmonic' initialization, we find that initialization is much less important when learning directly from larger datasets consisting of words alone. Of the algorithms in this paper, one does 2.5% better with harmonic initialization, another does 5% worse, and the other two are insensitive to initialization.

The rest of the paper is organized as follows. In Section 2, we discuss previous grammar induction research, in Section 3 we present the particular model grammar we will use, in Section 4 we describe the inference algorithms, and in Section 5 we present our experimental results.

2 Background

Previous grammar induction work has used datasets with at most $50,000$ sentences. Fully-lexicalized models would struggle with data sparsity on such small datasets, so previous work has assumed input in either the form of part-of-speech (POS) tags (Klein and Manning, 2004; Headden III et al., 2009) or word representations trained on a large external corpus (Spitkovsky et al., 2011; Le and Zuidema, 2015).

Some previous work has moved towards learning from word strings directly. Bisk and Hockenmaier (2013) used combinatory categorial grammar (CCG) to learn syntactic dependencies from word strings. However, they initialise their model by annotating nouns, verbs, and conjunctions in the training set with atomic CCG categories using a dictionary, and so do not learn from words alone. Pate and Goldwater (2013) learned syntactic dependencies from word strings alone, but used sentences from the Switchboard corpus of telephone speech that had been selected for prosodic annotation and so were unusually fluent.

Kim and Mooney (2010), Börschinger et al. (2011), and Kwiatkowski et al. (2012), learned from word strings together with logical form representations of sentence meanings. While children have situational cues to sentence meaning, these cues are ambiguous, and it is difficult to represent these cues in a way that is not biased towards the actual sentences under consideration. We focus on the evidence for syntactic structure that can be obtained from word strings themselves.

Grammar induction directly from word strings is interesting for two reasons. First, this problem setting more closely matches the language acquisition task faced by an infant, who will not have access to POS tags or a corpus external to her experience. Second, this setting allows us to attribute behavior of grammar induction systems to the underlying model itself, rather than additional annotations made to the input. Approaches to grammar induction that involve replacing words with POS tags or other lexical or syntactic observed labels make the process significantly more difficult to understand or compare across genres or languages, as the results will depend on exactly how these labels are assigned. Models that only require words alone as input do not suffer from this weakness.

3 PCFGs and the Dependency Model with Valence

3.1 Probabilistic Context Free Grammars

A Probabilistic Context Free Grammar is a tuple $(\boldsymbol{W}, \boldsymbol{N}, S, \boldsymbol{R}, \boldsymbol{\theta})$, where $\boldsymbol{W}$ and $\boldsymbol{N}$ are sets of terminal and non-terminal symbols, $S \in \boldsymbol{N}$ is a distinguished start symbol, and $\boldsymbol{R}$ is a set of production rules. $\boldsymbol{\theta}$ is a vector of multinomial parameters of length $|\boldsymbol{R}|$ indexed by production rules $A \to \beta$, so $\boldsymbol{\theta}_{A\to\beta}$ is the probability of the production $A \to \beta$. We use $\boldsymbol{R}_A$ to denote all rules with left-hand side A, and use $\boldsymbol{\theta}_A$ to denote the subvector of $\boldsymbol{\theta}$ indexed by the rules in $\boldsymbol{R}_A$. We require for all rules, $\theta_{A\to\beta} \geq 0$, and for all $A \in \boldsymbol{N}$, $\sum_{A\to\beta\in\boldsymbol{R}_A} \theta_{A\to\beta} = 1$, and use Δ to denote the probability simplex satisfying these constraints. The *yield* $\mathrm{y}(t)$ of a tree t is the string of terminals of t, and the yield of a vector of trees $\boldsymbol{T} = (t_1, \ldots, t_{|\boldsymbol{T}|})$ is the vector of yields of each tree: $\mathrm{y}(\boldsymbol{T}) = (\mathrm{y}(t_1), \ldots, \mathrm{y}(t_{|\boldsymbol{T}|}))$. The probability of generating a tree t given parameters $\boldsymbol{\theta}$ is:

$$P_G(t|\boldsymbol{\theta}) = \prod_{A\to\beta\in\boldsymbol{R}} \theta_r^{f(t,A\to\beta)}$$

where $f(t, A \to \beta)$ is the number of times rule $A \to \beta$ is used in the derivation of t.

To model uncertainty in the parameters, we draw the parameters of each multinomial $\boldsymbol{\theta}_A$ from a prior distribution $P(\boldsymbol{\theta}_A|\boldsymbol{\alpha}_A)$, where the vector of hyperparameters $\boldsymbol{\alpha}_A$ defines the shape of this prior distribution (and $\boldsymbol{\alpha}$ is just the concatenation of each $\boldsymbol{\alpha}_A$). The joint probability of a vector of trees $\boldsymbol{T}$ with one tree t_i for each sentence s_i, and parameters $\boldsymbol{\theta}$ is then:

$$P(\boldsymbol{T}, \boldsymbol{\theta}|\boldsymbol{\alpha}) = P(\boldsymbol{T}|\boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \prod_{A\to\beta\in\boldsymbol{R}} \theta_{A\to\beta}^{f(\boldsymbol{T},A\to\beta)} \left[\prod_{A\in\boldsymbol{N}} P(\boldsymbol{\theta}_A|\boldsymbol{\alpha}_A)\right]$$

where $f(\boldsymbol{T}, r)$ is the number of times rule r is used in the derivation of the trees in $\boldsymbol{T}$.

Dirichlet priors for these multinomials are both standard and convenient:

$$P_D(\boldsymbol{\theta}_A|\boldsymbol{\alpha}_A) = \frac{\Gamma\left(\sum_{A\rightarrow\beta\in\boldsymbol{R}_A}\alpha_{A\rightarrow\beta}\right)}{\prod_{A\rightarrow\beta\in\boldsymbol{R}_A}\Gamma(\alpha_{A\rightarrow\beta})}\prod_{A\rightarrow\beta\in\boldsymbol{R}_A}\theta_{A\rightarrow\beta}^{\alpha_{A\rightarrow\beta}-1}$$

where the Gamma function Γ generalizes the factorial function from integers to real numbers. Dirichlet distributions are convenient priors because they are *conjugate* to multinomial distributions: the product of a Dirichlet distribution and a multinomial distribution is itself a Dirichlet distribution:

$$P(\boldsymbol{T},\boldsymbol{\theta}|\boldsymbol{\alpha}) = P_G(\boldsymbol{T}|\boldsymbol{\theta})P_D(\boldsymbol{\theta}|\boldsymbol{\alpha}) \propto \prod_{A\rightarrow\beta\in\boldsymbol{R}}\theta_{A\rightarrow\beta}^{f(\boldsymbol{T},A\rightarrow\beta)+\alpha_{A\rightarrow\beta}-1} \tag{1}$$

For grammar induction, we observe only the corpus of sentences $\boldsymbol{C}$, and modify Equation 1 to marginalize over trees and rule probabilities.

$$P(\boldsymbol{C}|\boldsymbol{\alpha}) = \sum_{\boldsymbol{T}:\mathrm{y}(\boldsymbol{T})=\boldsymbol{C}}\int_{\Delta}P(\boldsymbol{T},\boldsymbol{\theta}|\boldsymbol{\alpha})d\boldsymbol{\theta} \tag{2}$$

This sum over trees introduces dependencies that make exact inference intractable.

We assessed grammar induction from words alone using the Dependency Model with Valence (DMV) (Klein and Manning, 2004). In the original presentation, it first draws the root of the sentence from a P_{root} distribution over words, and then generates the dependents of head h in each direction $dir \in \{\leftarrow,\rightarrow\}$ in a recursive two-step process. First, it decides whether to stop generating (a Stop decision) according to $P_{stop}(\cdot|h, dir, v)$, where v indicates whether or not h has any dependents in the direction of dir. If it does not stop (a $\neg$Stop decision), it draws the dependent word d from $P_{choose}(d|h, dir)$. Generation ceases when all words stop in both directions.

Johnson (2007) and Headden III et al. (2009) reformulated this generative process as a *split-head bilexical PCFG* (Eisner and Satta, 2001) so that the rule probabilities are DMV parameters. Such a PCFG represents each token of the string with two 'directed' terminals that handle leftward and rightward decisions independently, and defines rules and non-terminal symbols schematically in terms of terminals. Minimally, we need rightward-looking R_w, leftward-looking L_w, and undirected Y_w non-terminal labels for each word w. The grammar has a rule for each dependent word d of a head word h from the left ($L_h \rightarrow Y_d\ L_h$) and from the right ($R_h \rightarrow R_h\ Y_d$), a rule for each a word w to be the sentence root ($S \rightarrow Y_w$), and a rule for each undirected symbol to split into directed symbols ($Y_w \rightarrow L_w\ R_w$).

To incorporate Stop decisions into the grammar, we distinguished non-terminals that dominate a Stop decision from those that dominate a Choose decision by decorating Choose non-terminals with $'$ (so a left attachment rule is $L'_h \rightarrow Y_d\ L_h$), and introduced unary rules that rewrite to terminals ($L_h \rightarrow h_l$) for Stop decisions, and to Choose non-terminals ($L_h \rightarrow L'_h$) for $\neg$Stop decisions. We implemented valence with a superscript decoration on each non-terminal label: L^0_h indicates h has no dependents to the left, and L_h indicates that h has at least one dependent to the left. Figure 1 presents PCFG rule schemas with their DMV parameters, and dependency and split-head PCFG trees for "dogs bark." We use several inference algorithms to learn production weights for this PCFG, and study how the parsing accuracy varies with algorithm and computational effort.

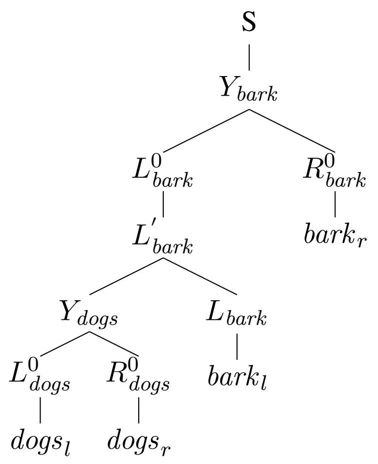
4 Inference[1]

One central challenge of learning from words alone is data sparsity. Data sparsity is most naturally addressed by learning from large amounts of data, which is easily available when learning from words alone, so we use algorithms that scale to large datasets. To ensure that our system reflects the underlying relationship between the model and the data, we explore three such algorithms. These algorithms are extensions of the batch algorithm for variational Bayesian (batch VB) inference of PCFGs due to Kurihara

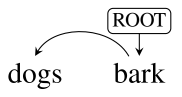
[1]Implementations and pre-processing software are available at `http://github.com/jkpate/streamingDMV`

PCFG Rule	DMV parameter
$S \rightarrow Y_h$	$P_{root}(h)$
$Y_h \rightarrow L_h^0 \; R_h^0$	1
$L_h^0 \rightarrow h_l$	$P_{stop}(\text{Stop}\|h, \leftarrow, \text{no\_dep})$
$L_h^0 \rightarrow L_h^{'}$	$P_{stop}(\neg\text{Stop}\|h, \leftarrow, \text{no\_dep})$
$L_h^{'} \rightarrow Y_d \; L_h$	$P_{choose}(d\|h, \leftarrow)$
$L_h \rightarrow h_l$	$P_{stop}(\text{Stop}\|h, \leftarrow, \text{one\_dep})$
$L_h \rightarrow L_h^{'}$	$P_{stop}(\neg\text{Stop}\|h, \leftarrow, \text{one\_dep})$

(a) Split-head rule schemas and corresponding probabilities for the DMV. The rules expanding L_h^0 and L_h symbols encode Stop decisions with no dependents and at least one dependent, respectively, and the the rules expanding L_h' symbols encode Choose decisions.



(b) Tree for "dogs bark" using the grammar in Figure 1a.



(c) Example dependency tree with one root and left arc.

Figure 1: The DMV as a PCFG, and dependency and split-head bilexical CFG trees for "dogs bark."

and Sato (2004), so we first review batch VB. Inspired by the reduction of LDA inference to PCFG inference presented in Johnson (2010), we then develop new streaming on-line PCFG inference algorithms by generalising the streaming VB (Broderick et al., 2013) and stochastic VB (Hoffman et al., 2010) algorithms for Latent Dirichlet Allocation (LDA) to PCFG inference. We finally review the collapsed VB algorithm due to Wang and Blunsom (2013) for PCFGs that we compare to the other algorithms.

Figure 2 summarizes the four algorithms for PCFG inference.

4.1 Batch VB

Kurihara and Sato's (2004) batch algorithm for variational Bayesian inference approximates the posterior $P(\boldsymbol{T}, \boldsymbol{\theta}|\boldsymbol{C}, \boldsymbol{\alpha})$ by maximizing a lower bound on the log marginal likelihood of the observations $\ln P(\boldsymbol{C}|\boldsymbol{\alpha})$. This lower bound $\mathcal{L}$ involves a variational distribution $Q(\boldsymbol{T}, \boldsymbol{\theta})$ over unobserved variables $\boldsymbol{T}$ and $\boldsymbol{\theta}$. By Jensen's inequality, for any distribution $Q(\boldsymbol{T}, \boldsymbol{\theta})$, we have:

$$\ln P(\boldsymbol{C}|\boldsymbol{\alpha}) = \ln \sum_{\boldsymbol{T}} \int Q(\boldsymbol{T}, \boldsymbol{\theta}) \frac{P(\boldsymbol{C}, \boldsymbol{T}, \boldsymbol{\theta}|\boldsymbol{\alpha})}{Q(\boldsymbol{T}, \boldsymbol{\theta})} d\boldsymbol{\theta} \geq \sum_{\boldsymbol{T}} \int Q(\boldsymbol{T}, \boldsymbol{\theta}) \ln \frac{P(\boldsymbol{C}, \boldsymbol{T}, \boldsymbol{\theta}|\boldsymbol{\alpha})}{Q(\boldsymbol{T}, \boldsymbol{\theta})} d\boldsymbol{\theta} = \mathcal{L}$$

$\ln P(\boldsymbol{C}|\boldsymbol{\alpha}) - \mathcal{L}$ is the Kullback-Leibler divergence KL $(Q(\boldsymbol{T}, \boldsymbol{\theta}) || P(\boldsymbol{T}, \boldsymbol{\theta}|\boldsymbol{C}, \boldsymbol{\alpha}))$. Variational inference adjusts the parameters of the variational distribution to maximize $\mathcal{L}$, which minimizes the KL divergence.

VB makes inference tractable by factorizing the variational posterior. The mean-field factorization assumes parameters and trees are independent: $Q(\boldsymbol{T}, \boldsymbol{\theta}) = Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \prod_{i=1}^{|\boldsymbol{T}|} Q_{\boldsymbol{T}}(t_i)$. Kurihara and Sato showed that $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is also a product of Dirichlet distributions, whose hyperparameters $\hat{\boldsymbol{\alpha}}_A$ are a sum of the prior hyperparameters $\alpha_{A \rightarrow \beta}$ and the expected count of $A \rightarrow \beta$ across the corpus under $Q_{\boldsymbol{T}}$:

$$\hat{\alpha}_{A \rightarrow \beta} = \alpha_{A \rightarrow \beta} + \sum_{i=1}^{|\boldsymbol{C}|} \hat{f}(s_i, A \rightarrow \beta)$$
$$\hat{f}(s, A \rightarrow \beta) = \mathbb{E}_{Q_{\boldsymbol{T}}} [f(t, A \rightarrow \beta)]$$

$f(t, A \rightarrow \beta)$ is the number of times rule $A \rightarrow \beta$ is used in the derivation of tree t. $\hat{f}(s_i, A \rightarrow \beta)$ is the expected number of times $A \rightarrow \beta$ is used in the derivation of sentence s_i, and can be computed using the Inside Outside algorithm (Lari and Young, 1990). Batch VB alternates between optimizing $Q_{\boldsymbol{\theta}}$, using expected counts, and $Q_{\boldsymbol{T}}$, using the hyperparameters $\hat{\boldsymbol{\alpha}}$ of $Q_{\boldsymbol{\theta}}$ to compute probability-like ratios $\pi_{A\to\beta}$:

$$Q_{\boldsymbol{T}}(t) = \prod_{A\to\beta\in\boldsymbol{R}} \pi_{A\to\beta}^{f(t,A\to\beta)} \qquad \pi_{A\to\beta} = \frac{\exp\left(\Psi\left(\hat{\alpha}_{A\to\beta}\right)\right)}{\exp\left(\Psi\left(\sum_{A\to\beta'\in\boldsymbol{R}_A}\hat{\alpha}_{A\to\beta'}\right)\right)}$$

where the digamma function $\Psi(\cdot)$ is the derivative of the log Gamma function. Algorithm 1 presents the full algorithm.

4.2 Scalable VB algorithms

Batch VB requires a complete parse of the training data before parameter updates, which is computationally intensive. We explore three algorithms that divide the data into minibatches $\boldsymbol{C} = \left\{\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n)}\right\}$ and update parameters after parsing each minibatch.

Streaming VB: Broderick et al. (2013) proposed a 'streaming VB' algorithm for LDA that approximates Bayesian Belief Updates (BBU) to make a single pass through the training data. A BBU uses the current posterior as a prior to compute the next posterior without reanalyzing previous minibatches:

$$P\left(\boldsymbol{\theta}|\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n)}\right) \propto P\left(\boldsymbol{C}^{(n)}|\boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}|\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n-1)}\right)$$

However, the normalization constant involves an intractable marginalization. Broderick et al. suggested approximating each posterior with some algorithm $\mathcal{A}$ that computes an approximate posterior $Q^{(n)}$ given a minibatch $\boldsymbol{C}^{(n)}$ and the previous posterior $Q^{(n-1)}$:

$$P\left(\boldsymbol{\theta}|\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n)}\right) \approx Q^{(n)}(\boldsymbol{\theta}) = \mathcal{A}\left(\boldsymbol{C}^{(n)}, Q^{(n-1)}(\boldsymbol{\theta})\right)$$

where $Q^{(0)}$ is the true prior. By using a mean-field VB algorithm for LDA inference for $\mathcal{A}$, they approximate each subsequent $Q^{(n)}$ as a product of Dirichlets, whose hyperparameters are a running sum of expected counts from previous minibatches and prior hyperparameters. We used the batch VB algorithm for $\mathcal{A}$ to generalise this algorithm to PCFG inference. Algorithm 2 presents the full algorithm.

Stochastic VB: Hoffman et al. (2010) proposed a 'stochastic VB' algorithm for LDA that uses each minibatch to compute the maximum of an estimate of the natural gradient of $\mathcal{L}$. This maximum is obtained by computing expected counts for $\boldsymbol{C}^{(i)}$, and scaling the counts as though they were gathered from the full dataset. The new hyperparameters are obtained by taking a step toward the maximum:

$$\alpha_{A\to\beta}^{(en+i)} = (1-\eta)\,\alpha_{A\to\beta}^{(en+i-1)} + \eta l_{A\to\beta}^{(i)} \hat{f}^{(en+i)}(A\to\beta)$$

where η is the step size, $\hat{f}^{(en+i)}(A\to\beta)$ is the expected count of rule $A\to\beta$ in minibatch i of epoch e, and $l_{A\to\beta}^{(i)}$ is the scaling term for rule $A\to\beta$. In their LDA inference procedure, each word has one topic, so the scaling term is the number of words in the full dataset divided by the number of words in the minibatch. For the DMV, a string s with $|s|$ terminals has one root, $|s|-1$ choose rules, and $2|s|+(|s|-1)$ stop decisions (two Stops and one ¬Stop rule for each arc). The scaling terms are then:

root rules	choose rules	stop rules
$l_{S\to Y_h}^{(i)} = \frac{\lvert\boldsymbol{C}\rvert}{\lvert\boldsymbol{C}^{(i)}\rvert}$	$l_{A\to\beta}^{(i)} = \frac{\sum_{j=1}^{\lvert\boldsymbol{C}\rvert}\lvert s_j\rvert-1}{\sum_{j'=1}^{\lvert\boldsymbol{C}^{(i)}\rvert}\lvert s_{j'}\rvert-1}$	$l_{A\to\beta}^{(i)} = \frac{\sum_{j=1}^{\lvert\boldsymbol{C}\rvert}2\lvert s_j\rvert+\lvert s_j\rvert-1}{\sum_{j'=1}^{\lvert\boldsymbol{C}^{(i)}\rvert}2\lvert s_{j'}\rvert+\lvert s_{j'}\rvert-1}$

Collapsed VB: Teh et al. (2007) proposed, and Asuncion et al. (2009) simplified, a 'collapsed VB' algorithm for LDA that integrates out model parameters and so achieves a tighter lower bound on the marginal likelihood. This algorithm cycled through the training set and optimized variational distributions over the topic assignment of each word given all the other words.

Wang and Blunsom (2013) generalized this algorithm to PCFGs. The variational distribution for each sentence is parameterized by expected rule counts for that sentence, and they optimize each sentence-specific distribution by cycling through the corpus and optimizing the distribution over trees for sentence s_i using counts from all the other sentences $\boldsymbol{C}^{(\neg i)}$. The exact optimization, marginalizing over rule probabilities, is intractable, so they instead use the posterior mean. Algorithm 4 presents the full algorithm.

```
    Data: a corpus of strings $\boldsymbol{C}$
    Initialization: prior hyperparameters $\boldsymbol{\alpha}$
1   $\hat{\boldsymbol{\alpha}}^{(0)} = \boldsymbol{\alpha}$
2   for $j = 1$ to $m$ do
3       $\pi_{A\to\beta} = \frac{\exp\left(\Psi\left(\hat{\alpha}^{(j-1)}_{A\to\beta}\right)\right)}{\exp\left(\Psi\left(\sum_{A\to\beta'\in\boldsymbol{R}_A}\hat{\alpha}^{(j-1)}_{A\to\beta'}\right)\right)}$
4       for $i = 1$ to $|\boldsymbol{C}|$ do
5           $\hat{f}^{(j)}(s_i, A \to \beta) = \mathbb{E}_{\boldsymbol{\pi}}\left[f(t, A \to \beta)\right]$
6       end
7       $\hat{\boldsymbol{\alpha}}^{(j)} = \boldsymbol{\alpha} + \hat{\boldsymbol{f}}^{(j)}$
8   end
    output: $\hat{\boldsymbol{\alpha}}^{(m)}$
```

Algorithm 1: Batch VB. Here, $\hat{\alpha}^{(j)}$ are the posterior counts after iteration j, which define rule weights π for the next iteration.

```
    Data: $n$ minibatches $\left\{\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n)}\right\}$
    Initialization: prior hyperparameters $\boldsymbol{\alpha}$
1   $\hat{\boldsymbol{\alpha}}^{(0)} = \boldsymbol{\alpha}$
2   for $i = 1$ to $n$ do
3       $\forall A \to \beta \in \boldsymbol{R}\ \hat{f}^{(0)}(A \to \beta) = 0$
4       for $j = 1$ to $m$ do
5           $\pi_{A\to\beta} = \frac{\exp\left(\Psi\left(\hat{f}\left(\boldsymbol{C}^{(i)}, A\to\beta\right)+\hat{\alpha}_{A\to\beta}\right)\right)}{\exp\left(\Psi\left(\sum_{A\to\beta'\in\boldsymbol{R}_A}\hat{f}\left(\boldsymbol{C}^{(i)}, A\to\beta'\right)+\hat{\alpha}_{A\to\beta'}\right)\right)}$
6           $\hat{f}^{(i,j)}(A \to \beta) = \mathbb{E}_{\boldsymbol{\pi}}\left[f(t, A \to \beta)\right]$
7       end
8       $\hat{\boldsymbol{\alpha}}^{(i)} = \hat{\boldsymbol{f}}^{(i,m)} + \hat{\boldsymbol{\alpha}}^{(i-1)}$
9   end
    output: $\hat{\boldsymbol{\alpha}}^{(n)}$
```

Algorithm 2: Streaming VB with m steps of VB per minibatch. $\hat{f}^{(j)}(A \to \beta)$ is the expected count of rule $A \to \beta$ in the i^{th} minibatch after j iterations.

```
    Data: $n$ minibatches $\left\{\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n)}\right\}$
    Initialization: prior hyperparameters $\boldsymbol{\alpha}$,
                    step size schedule parameters
                    $\tau$, $\kappa$, epoch count $E$
1   $\hat{\boldsymbol{\alpha}}^{(0)} = \boldsymbol{\alpha}$
2   for $e = 0$ to $E - 1$ do
3       for $i = 1$ to $n$ do
4           $\pi_{A\to\beta} = \frac{\exp\left(\Psi\left(\hat{\alpha}^{(en+i-1)}_{A\to\beta}\right)\right)}{\exp\left(\Psi\left(\sum_{A\to\beta'\in\boldsymbol{R}_A}\hat{\alpha}^{(en+i-1)}_{A\to\beta'}\right)\right)}$
5           $\hat{f}^{(en+i)}(A \to \beta) = \mathbb{E}_{\boldsymbol{\pi}}\left[f(t, A \to \beta)\right]$
6           $\eta = (\tau + i)^{-\kappa}$
7           for $A \to \beta \in \boldsymbol{R}$ do
8               $\hat{\alpha}^{(en+i)}_{A\to\beta} = (1 - \eta)\,\hat{\alpha}^{(en+i-1)}_{A\to\beta} + \eta l^{(i)}_{A\to\beta}\hat{f}^{(en+i)}(A \to \beta)$
9           end
10      end
11  end
    output: $\hat{\boldsymbol{\alpha}}^{(n)}$
```

Algorithm 3: Stochastic VB. $\hat{f}^{(en+i)}(A \to \beta)$ is the expected count of rule $A \to \beta$ in the i^{th} minibatch in the e^{th} epoch, and $l^{(i)}_{A\to\beta}$ is the scaling parameter for rule $A \to \beta$ for the i^{th} minibatch, as described in the text.

```
    Data: $n$ single-string minibatches
          $\left\{\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(n)}\right\}$
    Initialization: prior hyperparameters $\boldsymbol{\alpha}$,
                    epoch count $E$, initial
                    sentence-specific expected
                    counts $\hat{\boldsymbol{f}}$
1   $\hat{\boldsymbol{\alpha}} = \boldsymbol{\alpha} + \sum_{i=1}^{n}\hat{\boldsymbol{f}}^{(i)}$
2   for $e = 0$ to $E - 1$ do
3       for $i = 1$ to $n$ do
4           $\hat{\boldsymbol{\alpha}} = \hat{\boldsymbol{\alpha}} - \hat{\boldsymbol{f}}^{(i)}$
5           $\pi_{A\to\beta} = \frac{\hat{\alpha}_{A\to\beta}}{\sum_{A\to\beta'\in\boldsymbol{R}_A}\hat{\alpha}_{A\to\beta'}}$
6           $\hat{f}^{(i)}(A \to \beta) = \mathbb{E}_{\boldsymbol{\pi}}\left[f(t, A \to \beta)\right]$
7           $\hat{\boldsymbol{\alpha}} = \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{f}}^{(i)}$
8       end
9   end
    output: sentence-specific expected counts $\hat{\boldsymbol{f}}$,
            global hyperparameters $\hat{\boldsymbol{\alpha}}$
```

Algorithm 4: Collapsed VB. $\hat{f}^{(i)}(A \to \beta)$ is the expected count of rule $A \to \beta$ for the i^{th} sentence, and the global hyperparameters $\hat{\boldsymbol{\alpha}}$ are the sum of the expected counts for each sentence and prior hyperparameters.

Figure 2: The four variational Bayes algorithms for PCFG inference that are evaluated in this paper. Algorithm 1 is from Kurihara and Sato (2004), and Algorithm 4 is from Wang and Blunsom (2013). Algorithms 2 and 3 are novel PCFG inference algorithms developed here that generalise the LDA inference algorithms of Broderick et al. (2013) and Hoffman et al. (2010).

		train	dev	test
Swbd	Words	363,902	24,015	23,872
	Sentences	43,577	2,951	2,956
Fisher	Words	5,576,173	–	–
	Sentences	664,346	–	–

Table 1: Data set sizes. Fisher is only for training.

5 Experiments

We evaluate how millions of words of training data affects grammar induction from words alone by examining learning curves. We ran each algorithm 5 times, each with a different random shuffle of the training data on each run, and evaluated on the test set at logarithmically-spaced numbers of training sentences. Stochastic, collapsed, and batch VB used more than one pass over the training corpus, while streaming VB makes one pass over the training corpus. To obtain a consistent horizontal axis in our learning curves, we plot learning curves as a function of computational effort, which we measure by the number of sentences parsed, since almost all the computational effort of all the algorithms is in parsing.

Stochastic, collapsed, and streaming VB can learn from the full training corpus (although collapsed VB requires more RAM – 60GB rather than 6GB for us – as it stores expected rule counts for each sentence). Batch VB required about 50 iterations for convergence for large training sets, and so cannot be applied to the full training set due to long training times. We used batch VB with training sets of up to $100,000$ strings.

5.1 Hyperparameters and initialization

We use Dirichlet priors with symmetric hyperparameter $\alpha = 1$ for all algorithms (preliminary experiments showed that the algorithms are generally insensitive to hyper-parameter settings). We ran batch VB until the log probability of the training set changed less than 0.001%. For stochastic VB, we used $\kappa = 0.9$, $\tau = 1$, and minibatches of 10,000 sentences. To investigate convergence and overfitting, we ran stochastic and collapsed VB for 15 epochs of random orderings of the training corpus. For streaming VB, the first minibatch had $10,000$ sentences, the rest had 1, and we used one iteration of VB per minibatch.

Klein and Manning (2004) showed that initialization strongly influences the quality of the induced grammar when training from POS-tagged WSJ10 data, and they proposed a harmonic initialization procedure that puts more weight on rules that involve terminals that frequently appear close to each other in the training data. We present results both for a uniform initialization, where the only counts initially are the uninformative Dirichlet priors (plus, for collapsed VB, random sentence-specific counts), and a harmonic initialization. For streaming VB, harmonic counts are gathered from each minibatch, and for the others, harmonic counts are gathered from the entire training set.

5.2 Data

We present experiments on two corpora of words from spontaneous speech transcripts. Our first corpus is drawn from the Switchboard portion of the Penn Treebank (Calhoun et al., 2010; Marcus et al., 1993). We used the version produced by Honnibal and Johnson (2014), who used the Stanford dependency converter to convert the constituency annotations to dependency annotations (Catherine de Marneffe et al., 2006). We used Honnibal and Johnson's train/dev/test partition, and ignored their second dev partition. We discarded sentences shorter than four words from all partitions, as they tend to be formulaic backchannel responses (Bell et al., 2009), and sentences with more than 15 words (long sentences did not improve accuracy on the dev set).

We augmented the Switchboard training set with the Fisher corpus of telephone transcripts. We again used only sentences with 4 to 15 words. Unlike the words-only evaluation of Pate and Goldwater (2013), which used only the fluent sentences from Switchboard that had been prosodically annotated, both of these corpora contain disfluencies. These corpora have a vocabulary of $40,597$ word types. The grammars have one Root rule for each word type, four Stop rules (two directions x two Stop decisions) for

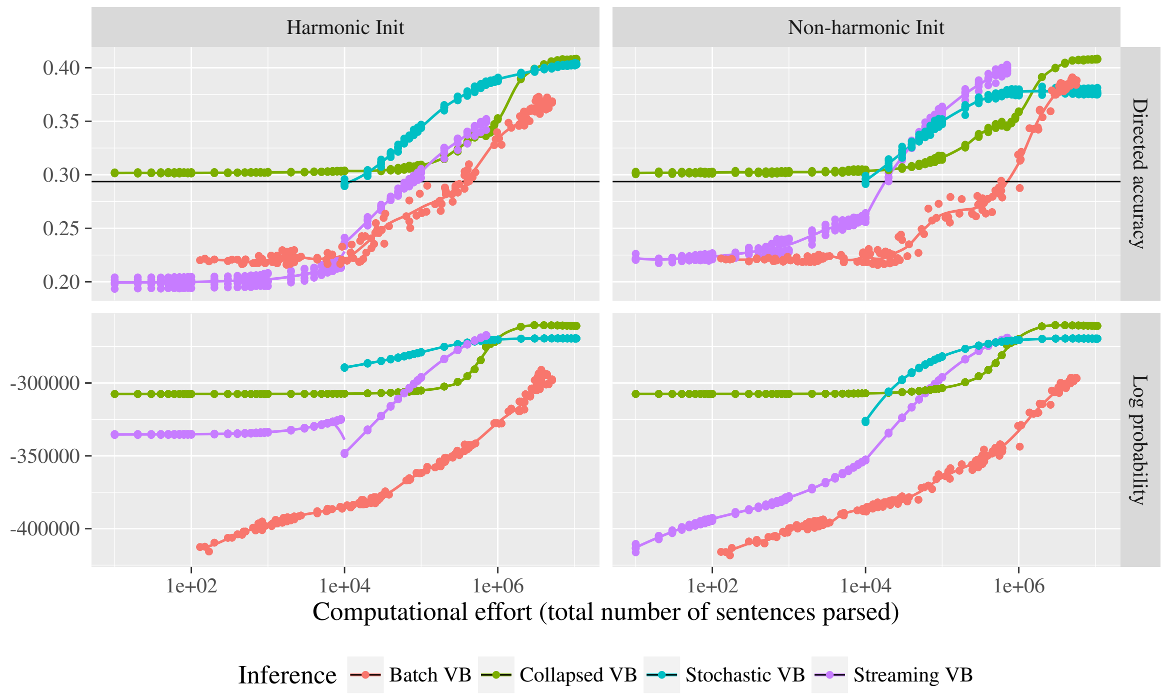


Figure 3: Directed accuracy (top) and predictive log probability (bottom) of test-set sentences from Switchboard with 4-15 words. The horizontal axis is the number of sentences parsed (all algorithms except streaming VB re-parse sentences multiple times). The left column presents inference with a harmonic initialization, and the right column presents inference with a uniform initialization (and, for collapsed VB, random sentence-specific counts). The black line is a uniform-branching baseline.

each word type, and $5,381,644$ Choose rules. Table 1 presents data set sizes.

5.3 Evaluation measures

We evaluated all algorithms in terms of predictive log probability and directed attachment accuracy. We computed the log probability of the evaluation set under posterior mean parameters, obtained by normalizing counts. Directed accuracy is the proportion of arcs in the Viterbi parse that match the gold standard, including the root. We also compared with a left-branching baseline, since it outperformed a right-branching baseline. A left-branching (right-branching) baseline sets the last (first) word of each sentence to be the root, and assigns each word to be the head of the word to its left (right). The left-branching baseline on this dataset is about 0.29, while on the traditional `wsj10` dataset of Klein and Manning (2004) it is 0.336, suggesting our dataset, with longer sentences, is somewhat more difficult.

5.4 Results

The bottom row of Figure 3 presents the predictive log probability of the test set under the posterior mean parameter vector as training proceeds. The figure contains one point per evaluation per run, and a loess-smoothed curve for each inference type. We see among all algorithms that the log probability of the test set constantly increases, regardless of initialization, with one exception. The sole exception is streaming VB with harmonic initialization, where predictive log probability drops after the initial minibatch of 10,000 sentences. Streaming VB with harmonic init parses each sentence of the initial minibatch using prior pseudocounts and harmonic counts. We will return to this drop when we discuss accuracy.

Batch VB learns more slowly (as a function of computational effort) than the online and minibatch algorithms, but the online and minibatch algorithms all ultimately obtain similar performance. Collapsed VB obtains the best predictive log probability, which, as it integrates out parameters and therefore has a tighter bound, is to be expected (Teh et al., 2007). There is no clear advantage to the harmonic initialization except early in training for streaming and stochastic VB, so it may be that earlier results showing the importance of harmonic initialisation reflect the small training data sets used in those experiments.

The top row of Figure 3 presents directed accuracy on the test set as training proceeds. As in the predictive probability evaluation, there is no clear advantage to a harmonic initialization across algorithms. Batch VB and collapsed VB perform identically with both initializations, and streaming VB ultimately does 5% better while stochastic VB does 2.5% worse. While streaming VB showed a drop in predictive probability after the initial 10,000 sentence minibatch with harmonic initialization, it obtains a small but sharp improvement in parse accuracy at the same point. These two results suggest that the harmonic initialization, applied to words, captures regularities that are not syntactic but still explain data well.

The inference algorithms differ most obviously when they have parsed few sentences, indicating that each algorithm's bias is strongest in the face of small data. Streaming VB learns slowly initially because, throughout the large initial minibatch, it gathers counts using only the uninformative prior or only the uninformative prior plus harmonic counts. Collapsed VB, on the other hand, has sentence-specific counts for the entire training corpus even in the random case. These counts provide a rough estimate of how many opportunities there are for an arc to exist between each word in each direction at each valence, and therefore provide a stronger starting point that takes more time to overcome. Finally, the good performance of stochastic VB with small datasets, compared to streaming VB and batch VB, may reflect the conservatism of only taking a step in the direction of the gradient rather than always maximizing.

Regardless of the details of the different algorithms' performance, we see that they all steadily improve or stabilize as inference proceeds over a large dataset, and that initialization is not important when learning from large numbers of words.

6 Conclusion

Grammar induction from words alone has the potential to address important questions about how children learn and represent linguistic structure, but previous work has struggled to learn from words alone in a principled way. Our experiments show that grammar induction from words alone is feasible with a simple and well-known model if the dataset is large enough, and that heuristic initialization is not necessary (and may even interfere). Future computational work on child language acquisition should take advantage of this finding by applying richer models of syntax to large datasets, and learning distributed word representations jointly with syntactic structure.

References

Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. 2009. On smoothing and inference for topic models. In *Uncertainty in Artificial Intelligence*.

Alan Bell, Jason M Brenier, Michelle Gregory, Cynthia Girand, and Dan Jurafsky. 2009. Predictability effects on durations of content and function words in conversational English. *Journal of Memory and Language*, 60:92–111.

Yonatan Bisk and Julia Hockenmaier. 2013. An HDP model for inducing Combinatory Categorial Grammars. *Transactions of the Association for Computational Linguistics*, 1(Mar):75–88.

Benjamin Börschinger, Bevan K Jones, and Mark Johnson. 2011. Reducing grounded learning tasks to grammatical inference. In *Proceedings of the 2011 conference on Emprical Methods in Natural Language Processing (EMNLP)*, pages 1416–1425.

Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michale I Jordan. 2013. Streaming variational bayes. In *Neural Information Processing Systems*.

S Calhoun, J Carletta, J Brenier, N Mayo, D Jurafsky, M Steedman, and D Beaver. 2010. The NXT-format Switchboard corpus: A rich resource for investigating the syntax, semantics, pragmatics and prosody of dialogue. *Language Resources and Evaluation*, 44(4):387–419.

Marie Catherine de Marneffe, Bill MacCartney, and Christopher D Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*.

Noam Chomsky. 1986. Knowledge of language: Its nature, origin, and use.

Jason Eisner and Giorgio Satta. 2001. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *ACL*.

Will Headden III, Mark Johnson, and David McClosky. 2009. Improved unsupervised dependency parsing with richer contexts and smoothing. In *NAACL-HLT*.

M Hoffman, D M Blei, and F Bach. 2010. Online learning for latent dirichlet allocation. In *Proceedings of NIPS*.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, 2(April):131–142.

Mark Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parseable CFGs with unfold-fold. In *ACL*.

Mark Johnson. 2010. PCFGs, topic models, adaptor grammars and learning topical collocations and the structure of proper names. In *Proceedings of the Association for Computational Linguistics*.

Joohyun Kim and Raymond J Mooney. 2010. Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd international conference on computational linguistics (COLING 2010)*.

Dan Klein and Christopher D. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of ACL*, pages 479–486.

Kenichi Kurihara and Taisuke Sato. 2004. An application of the Variational Bayesian approach to probabilistic context-free grammars. In *IJCNLP 2004 Workshop beyond Shallow Analyses*.

Tom Kwiatkowski, Sharon Goldwater, Luke Zettlemoyer, and Mark Steedman. 2012. A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. In *Proceedings of the European Chapter of the Association for Computational Linguistics*.

K Lari and S J Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 5:237–257.

Phong Le and Willem Zuidema. 2015. Unsupervised dependency parsing: Let's use supervised parsers. In *Proceedings of ACL*.

Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

John K Pate and Sharon Goldwater. 2013. Unsupervised dependency parsing with acoustic cues. *Transactions of the ACL*.

Valentin I Spitkovsky, Hiyan Alshawi, Angel X Chang, and Daniel Jurafsky. 2011. Unsupervised dependency parsing without gold part-of-speech tags. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*.

Yee Whye Teh, David Newman, and Max Welling. 2007. A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation. In *Neural Information Processing Systems*, volume 19, pages 705–729.

Pengyu Wang and Phil Blunsom. 2013. Collapsed variational bayesian inference for PCFGs. In *Proceedings the Seventeenth Conference on Computational Natural Language Learning*, pages 173–182.