A sampling and inference implementation for fragment grammars

Yves Blain-Montesano
McGill University
Cognitive Science Program
`yves.blain-montesano@mail.mcgill.ca`

1 Introduction

We describe in this technical report a Julia[1] implementation of fragment grammars (O'Donnell, 2015), a stochastically memoized, lazily unfolded generalization of probabilistic context-free grammars. We first describe the generative model and a method of inference in section 2. In section 3 we describe the implementation's data types and interface.

Intuitively, fragment grammars store work from previously generated derivation trees as contiguous tree fragments and allow to reuse them when deriving new trees. The formalism is naturally suited to computational linguistics, but is applicable wherever else probabilistic context-free grammars are.

2 Fragment grammars

2.1 Generative model

We say a CFG $\mathcal{G}$ is a tuple

$$\mathcal{G} = \langle V, T, R, S \rangle,$$

where V is the set of nonterminal symbols, T is the set of terminal symbols, R is the set of rules r of form $V^+ \rightarrow \{V \cup T\}^+$. We allow any element of this tuple to be indexed by its CFG $\mathcal{G}$.

Fragment grammars are a generalization of probabilistic context-free grammars (PCFGs) with a Dirichlet prior over rule probabilities for every nonterminal. As in an adaptor grammar (Johnson et al., 2007), there is a Pitman-Yor process associated with each nonterminal according to whose distribution derivation trees are stochastically memoized. A key addition to these features in fragment grammars is a beta-binomial distribution for each right-hand side nonterminal of a given rule according to which, at every recursive step of a derivation, we either stop or continue the recursion at a nonterminal.

We present this model using the same notation as O'Donnell (2015). Formally, a fragment grammar is a tuple

$$\mathcal{F} = \Big\langle \mathcal{G}, \{\vec{\pi}^{\mathtt{A}}\}_{\mathtt{A} \in V_{\mathcal{G}}}, \{\langle a^{\mathtt{A}}, b^{\mathtt{A}} \rangle\}_{\mathtt{A} \in V_{\mathcal{G}}}, \\ \{\vec{\psi}_{\mathtt{B}}\}_{\mathtt{B} \in \mathtt{rhs}(r \in R_{\mathcal{G}})} \Big\rangle,$$

where $\mathcal{G}$ is a context-free grammar (CFG), hereafter called the base grammar, $\{\vec{\pi}^{\mathtt{A}}\}_{\mathtt{A} \in V_{\mathcal{G}}}$ is the set of pseudocount vectors for each nonterminal's associated Dirichlet-multinomial distribution, and $\{\vec{\psi}_{\mathtt{B}}\}_{\mathtt{B} \in \mathtt{rhs}(r \in R_{\mathcal{G}})}$ is the set of binary pseudocount vectors for the beta-binomial distribution associated to each nonterminal on each rule's right-hand side.

We present the generative model at a glance before explaining it further. The following recursive stochastic equations describe the probability of a derivation tree under the fragment grammar model:

$$G^{\mathtt{a}}_{\mathtt{FG}}(d) = \begin{cases} \displaystyle\sum_{s \in \mathtt{prefix}(d)} & \Big[\mathrm{mem}\,\{L^{\mathtt{a}}\}\,(s) \\ & \prod_{i=1}^{n} G^{\mathtt{root}(s'_i)}_{\mathtt{FG}}\,(s'_i) \Big] \\ & \text{if } \mathtt{root}(d) = \mathtt{a} \in V_{\mathcal{G}}, \\ 1 & \text{if } \mathtt{root}(d) = \mathtt{a} \in T_{\mathcal{G}} \end{cases}$$

$$L^{\mathtt{A}}(d) = \sum_{\mathtt{A} \rightarrow \mathtt{root}(\hat{d}_i), \cdots, \mathtt{root}(\hat{d}_k)} \theta_r \prod_{i=1}^{k} \Big[\nu_{\mathtt{root}(\hat{d}_i)} \\ G^{\mathtt{root}(\hat{d}_i)}_{\mathtt{FG}} \left(\hat{d}_i\right) + \left(1 - \nu_{\mathtt{root}(\hat{d}_i)}\right) \Big]$$

[1] `julialang.org`

$$\vec{\theta}^{\mathtt{A}} \sim \text{DIRICHLET}\left(\vec{\pi}^{\mathtt{A}}\right)$$
$$\vec{\nu}_{\mathtt{B}} \sim \text{BETA-BIN}\left(\vec{\psi}_{\mathtt{B}}\right)$$
$$\text{mem}\{L^{\mathtt{A}}\} \sim \text{PYP}(a^{\mathtt{A}}, b^{\mathtt{A}}, L^{\mathtt{A}})$$

For a derivation tree d, $L^{\mathtt{A}}(d)$ represents the probability of using a rule r at the root of d and expanding with probability ν the current derivation to the rest of d below each right-hand side with probability given by $G^{\mathtt{a}}_{\mathtt{FG}}$, marginalized over all base grammar rules r from nonterminal $\mathtt{A}$ which produce on their right-hand side the children of the root node of d.

For a derivation tree d, $G^{\mathtt{a}}_{\mathtt{FG}}(d)$ represents the marginal probability (over all contiguous tree fragments whose roots are d's root—i.e. over its prefixes) of sampling a prefix of d with probability given by $\text{mem}\{L^{\mathtt{A}}\}$ multiplied by the probability $G^{\mathtt{a}}_{\mathtt{FG}}$ assigns to the derivation tree starting at the prefix's leaves.

An explanation of $\text{mem}\{L^{\mathtt{A}}\}$ follows a reminder of the Pitman-Yor process below.

The Pitman-Yor process (denoted PYP above) can be viewed as a distribution over distributions over a countably infinite support. It can also be viewed as a sequential partitioning of observations, the generalized Chinese restaurant process (Ishwaran and James, 2003). Under this analogy, each of a partition's disjoint sets is a table at a restaurant where the plate being served is the value of the observation sampled. For every observation, a customer seats themselves at the appropriate table. There are infinitely many tables, though they are only unveiled as needed. A customer sits at a new table with probability

$$\frac{Ka+b}{N+b},$$

and sits at the ith table of the restaurant with probability

$$\frac{y_i - a}{N+b},$$

where y_i is the number of customers at table i, N is the total number of customers, K is the number of tables currently occupied, $a \in [0,1]$ is called the discount parameter, and $b \in [-a, \infty[$ is the concentration parameter. Note that when $a = 0$, the process reduces to a Dirichlet distribution (Teh, 2006). Because all observations seated at a same table are of the same value, they can be reordered in any way. In fact, all observations entered into the partition can be reordered arbitrarily and retain the same information (Johnson et al., 2007). The number of observations at each table is a sufficient statistic for the joint probability of the restaurant[2] (O'Donnell, 2015).

The Pitman-Yor process demands a base distribution from which to sample observations which are then seated as customers. For fragment grammars, the base distribution is scored by $L^{\mathtt{A}}$ above. Undoing the marginalization in that equation, we see that the base distribution involves sampling a base grammar rule. Then, it either expands each child nonterminal with a recursively sampled nesting of tree fragments, or stops the current fragment at the child nonterminal and samples a new fragment from that position onward. While the resulting derivation tree is the same, the observed fragments unique to this subdivision of a derivation tree d differ based on precisely where a fragment stops expanding.

Because each of those observed fragments with some root node $\mathtt{A}$ is then "seated" within a Pitman-Yor restaurant associated with $\mathtt{A}$, a sample from that restaurant allows one to observe a preexisting fragment if this sample's new customer is seated at the relevant preexisting table.

Now it is intuitive to see $\text{mem}\{L^{\mathtt{A}}\}$ as a memoization of a fragment obtained according to $L^{\mathtt{A}}$, where we either obtain a novel fragment at a new table, or we obtain a reused fragment at a populated table.

We call the collection of PYP fragment observations, Dirichlet-multinomial rules, and beta-binomial observations sampled from the model for a derivation tree $d^{(i)}$ its analysis $f^{(i)}$. For a data set $\mathbf{D}$, the set of corresponding analyses is $\mathbf{F}$.

2.2 Markov chain Monte Carlo

O'Donnell (2015) describes a Metropolis-Hastings sampler, a Markov chain Monte Carlo (MCMC) method, which we implement accordingly.

We encourage the reader to examine Bishop's (2006) introduction to the method for further details.

MCMC is an inference method which approximates the posterior of a distribution by defining a transition kernel between hypothesis distributions. We therefore have a Markov chain where states are hypothesized posterior distributions. We

[2] The beta-binomial and Dirichlet-multinomial distributions, also being Pólya urn models, have the same property of exchangeability without independence (Helfand, 2013).

transition between them according to the kernel, whose long-term behavior converges near the optimal posterior distribution.

In the case of fragment grammars, which are exchangeable and for which the counts of the observations in $\mathbf{F}$ are a sufficient statistic (O'Donnell, 2015), any arbitrary $f^{(i)} \in \mathbf{F}$ can be considered the latest observation. The kernel therefore picks some data $d^{(i)}$'s analysis $f^{(i)}$, removes all observations associated with it to obtain state $\mathbf{F}^{-}$, and from it samples a new $f^{(i)+}$ from to obtain $\mathbf{F}^{+}$, the proposed distribution.

The Metropolis-Hastings criterion accepts the proposed distribution given by the kernel with probability

$$\min\left(1, \frac{p(\mathbf{F}^{+})}{p(\mathbf{F})} \cdot \frac{q(f^{(i)} \mid \mathbf{F}^{-})}{q(f^{(i)+} \mid \mathbf{F}^{-})}\right),$$

where p and q need not be normalized and therefore may not be probabilities (Bishop, 2006), but represent the probability of the hypothesis distribution described by $\mathbf{F}$ or $\mathbf{F}^{+}$, and the posterior probability of $f^{(i)}$ or $f^{(i)+}$ given $\mathbf{F}^{-}$, respectively. Hyperparameters are excluded for clarity.

However, because independence between observations does not hold for Pólya urn models, calculating $q(f^{(i)} \mid \mathbf{F}^{-})$ is intractable. O'Donnell (2015) defines an approximating PCFG whose rules represent fragments and base grammar rules, from which a parse can be obtained and scored. A rule $\rho^{\mathtt{A}}_{\gamma}$ which collapses fragments and base rules sharing the same left-hand side `A` and right-hand side γ[3] is added for every such case in $\mathcal{F}$ and $\mathcal{G}$ with probability

$$\theta_{\rho^{\mathtt{A}}_{\gamma}} = \sum_{\mathtt{v} \in \mathrm{mem}\{L^{\mathtt{A}}\} | \mathtt{yield(v)} = \gamma} \frac{y^{\mathtt{A}}_{\mathtt{V}} - a^{\mathtt{A}}}{N^{\mathtt{A}} + b^{\mathtt{A}}} + \sum_{r \in R^{\mathtt{A}}_{\mathcal{G}} | \mathtt{rhs}(r) = \gamma} \left[\frac{K^{\mathtt{A}} a^{\mathtt{A}} + b^{\mathtt{A}}}{N^{\mathtt{A}} + b^{\mathtt{A}}} \times \frac{x^{\mathtt{A}}_{r} + \pi^{\mathtt{A}}_{r}}{K^{\mathtt{A}} + \sum \vec{\pi}^{\mathtt{A}}} \right].$$

The inside probability is found for the forest of approximating parses for $d^{(i)}$ from which an approximating parse tree is sampled using span-wise conditioning (O'Donnell, 2015), i.e. ensuring that every constituent added during parsing is present in $d^{(i)}$. A fragment grammar analysis is found by sampling conditionally on the collapsed rules bottom-up.

[3]The RHS for a fragment is simply its leaves (its yield).

3 Implementation

3.1 Data types

Several data types are defined as part of the current implementation.

The `Fragment` type holds a `Tree` which represents a contiguous fragment of some derivation tree held in its field `tree`. It holds references to its nonterminal leaves in `variables`, and all leaves in `leaves`.

The `Pointer` type represents a sequence of fragments whose leaves are ultimately terminals. The `fragment` field holds the fragment under question, and `children` is a dictionary mapping `fragment.variables` to a set of `Pointer` objects. Traversing this in the order of `fragment.variables` yields the same derivation tree it was constructed from.

`Analysis` is a container for a fragment grammar parse and its associated observations. This is convenient for adding and removing observations.

```
struct Fragment
    tree ::Tree
    variables ::Vector{Tree}
    leaves ::Vector{Tree}
end

struct Pointer
    fragment ::Fragment
    children ::Dict{Tree, Pointer}
end

struct Analysis{C, CR}
    pointer ::Pointer
    dm_obs ::Vector{Tuple{C,CR}}
    bb_obs ::Vector{Pair{Tuple{C, CR, C},Bool}}
    crp_obs ::Vector{Fragment}
end
```

An abstract type `AbstractRule{C1,C2}` is the supertype of three rule types. `{C1,C2}` are the types of the LHS and RHS symbols of the rule type. This will be omitted below.

`BaseRule` is a regular CFG rule with any number of symbols on the RHS[4]. This type should be directly accessible by the user.

The `FragmentRule` type should only be used internally when adding observations or during inference. It contains a reference to a fragment whose root category is its LHS and whose leaves are its RHS.

An `ApproxRule` is used when parsing with the approximating PCFG. It contains base grammar rules and fragments that correspond to it. The

[4]That `Vararg` allows to have no RHS symbols is intended for compatibility.

probability fields are updated during inference as needed.

```
abstract type AbstractRule{C1,C2} end
lhs(r::AbstractRule)
rhs(r::AbstractRule)

struct BaseRule{C1,C2} <:AbstractRule{C1,C2}
    lhs ::C1
    rhs ::Tuple{Vararg{C2}}
end

struct FragmentRule{C1,C2} <:AbstractRule{C1,C2}
    fragment ::Fragment
    lhs ::C1
    rhs ::Tuple{Vararg{C2}}
end

mutable struct ApproxRule{C1,C2} <:AbstractRule{C1,
     C2}
    lhs ::C1
    rhs ::Tuple{Vararg{C2}}
    rules ::Vector{AbstractRule{C1,C2}}
    probs ::Vector{LogProb}
    prob ::LogProb
end
```

The `FragmentGrammar` type is parameterized by the type of its nonterminal symbols `C`, the type of its terminal symbols `T`, and the rule types for nonterminal rules and terminal rules `CR` and `TR`. It contains the base grammar's nonterminals in `categories`, a subset of which should be start symbols in `startcategories`, its terminal symbols, and its rules. `CRP`, `DM`, and texttt BB are dictionaries mapping indices to distributions, e.g., nonterminal symbols to Pitman-Yor processes (denoted CRP) and to Dirichlet-multinomials. Note that `BB` is indexed by a tuple of an LHS nonterminal, a rule, and a right-hand side nonterminal. The `startstate` and `terminal_dict` fields hold completions for parsing. The state represents the initial state of a finite state automaton used for completions during parsing (Harasim et al., 2017).

```
mutable struct FragmentGrammar{C, CR, T, TR}
    categories ::Vector{C}
    startcategories ::Vector{C}
    category_rules ::Vector{CR}
    terminals ::Vector{T}
    terminal_rules ::Vector{TR}
    preterminals ::Vector{C}
    startstate ::State{C, AbstractRule{C,C}}
    terminal_dict ::Dict{T, Vector{Tuple{C, TR}}}
    CRP ::Dict{C, ChineseRest{Fragment}}
    DM ::Dict{C, DirCat{CR, Float64}}
    BB ::Dict{Tuple{C, CR, C}, BetaBern{Bool, Int}}
end
```

3.2 Interface

Below are the methods presented to the end-user. The first is a constructor for `FragmentGrammar`. Parameters `dm_pseudo`, `bb_alpha`, and `bb_beta` are used to initialize all distributions with the same hyperparameters.

`run_mcmc!` takes a newly constructed, empty `FragmentGrammar`, a vector of input trees equivalent to $\mathbf{D}$, and a number of times it should iterate over the data set.

`sample` is a forward sampling method that returns a new fragment grammar derivation in the form of an object of type `Tuple{Pointer, Tuple{C,CR}[], Pair{Tuple{C,CR,C}, Bool}[], Fragment[]}`, from which one can construct an `Analysis`.

The methods `add_obs!` and `rm_obs!` add observations from objects of type `Analysis`. When a fragment is observed or removed, the finite state automaton's relevant state is updated: the `ApproxRule` corresponding to the fragment to has a `FragmentRule` added to it. The joint probability of the PYP is updated incrementally when observations are added or removed.

`logscore` returns the joint probability of the fragment grammar.

```
function FragmentGrammar(cats::Vector{C}, starts::
     Vector{C}, cat_rules::Vector{CR}, terms::Vector
     {T}, term_rules::Vector{TR}, a, b, dm_pseudo,
     bb_alpha, bb_beta) where {C, CR<:BaseRule{C,C},
      T, TR}

function run_mcmc!(fg::FragmentGrammar, inputs::
     Vector{Tree}, sweeps::Int)

function sample(fg ::FragmentGrammar{C,CR,T,TR}, cat
      ::C) where {C, CR, T, TR}

function add_obs!(fg ::FragmentGrammar, analysis ::
     Analysis)

function rm_obs!(fg ::FragmentGrammar, analysis ::
     Analysis)

function logscore(fg::FragmentGrammar)
```

3.3 Implementation details

Our implementation uses a package by Harasim et al. (2017) for parsing which we have modified slightly for our purpose, among other reasons to enable conditioning parses on constituent information of input derivations in $\mathbf{D}$. Their framework is an agenda-based semiring chart parser inspired by the likes of Shieber et al. (1995).

A few particularities arise out of a need for compatibility. Trees contained in fragments only yield preterminals at most. This is not a hindrance, as we can assume all terminals have unique preterminals. We can resolve a fragment grammar parse's preterminals to their respective terminals.

As mentioned above, a finite state automaton is updated to contain approximating PCFG rules which marginalize over rules incrementally, as observations are added. This allows for efficient sampling of an analysis from an approximating parse.

Currently, the initialization of the MCMC method initializes fragments of depth 1 for every rule application in the input data set instead of the batch initialization described in (O'Donnell, 2015). In addition to this, our MCMC method iterates over data in the same order without shuffling.

3.4 Example

Below is an example of inference for a data set of ten identical trees for a simple grammar.

```
fg =FragmentGrammar(["S"], ["S"], [BaseRule("S", ("S
    ", "T")), BaseRule("S", ("T", "S")), BaseRule("
    S", ("T",))], ["a"], [BaseRule("T", ("a",))], 0
    .5, 0.5, 1, 1)

test_trees =Tree[tree("[S[S[S[S[T[a]]][T[a]]][T[a
    ]]][T[a]]]") for i in 1:10]

analyses =run_mcmc!(fg, test_trees, 100)

logscore(fg)
```

4 Conclusion

We have presented an implementation in the Julia language capable of sampling, inference, and parsing of the fragment grammar formalism. We have described the formalism as well as practicalities involving the use of our implementation.

Acknowledgments

This work was produced for an honours thesis under the supervision of Timothy J. O'Donnell. Thanks to Daniel Harasim et al. (2017; 2018) for their generalized chart parsing package and distribution code. Thanks to Michaela Socolof for contributions and testing.

References

Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

Daniel Harasim, Chris Bruno, Eva Portelance, Martin Rohrmeier, and Timothy J. O'Donnell. 2017. A generalized parsing framework for abstract grammars.

Daniel Harasim, Martin Rohrmeier, and Timothy J. O'Donnell. 2018. A generalized parsing framework for generative models of harmonic syntax. In *ISMIR*.

Nora Helfand. 2013. Pólya's urn and the beta-bernoulli process.

Hemant Ishwaran and Lancelot F James. 2003. Generalized weighted chinese restaurant processes for species sampling mixture models. *Statistica Sinica*, pages 1211–1235.

Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. 2007. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. In *Advances in neural information processing systems*, pages 641–648.

Timothy J O'Donnell. 2015. *Productivity and reuse in language: A theory of linguistic computation and storage*. MIT Press.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36.

Yee Whye Teh. 2006. A hierarchical bayesian language model based on pitman-yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 985–992, Stroudsburg, PA, USA. Association for Computational Linguistics.