A Stochastic Memoizer for Sequence Data

Wood, Archambeau, Gasthaus, James, and Teh

Presented by: Will Allen

October 27, 2011

Quick Note

- This paper builds on the Teh's 2006 ACL article on PYP for language models presented on Tuesday.
- And relies on details about the coagulation and fragmentation operators in Gasthaus and Teh's 2010 NIPS article presented next.
- So I'll go over those topics relatively quickly and incompletely.

Main Idea

- *Problem:* Want to model sequences of symbols $\mathbf{x}_{1:T} = (x_1 x_2 ... x_T) \in \Sigma^*$, without making Markov assumptions. (Preferably maintaining power-law symbol occurrance statistics.)
- E.g.: Given some new symbol, x_{T+1}, we'd like to find $p(x_{T+1} = s|\mathbf{x}_{1:T}) \forall s \in \Sigma$.
- This requires a large (infinite) number of latent variables.
- *Solution:* Use a tree data structure, and clever use of marginalization, to efficiently represent a hierarchical Pitman-Yor process prior over the predictive distribution.

Markov Models

- Normally, when modeling language, we make a *Markov assumption*:
- Given sequence $x_{1:T} = (x_1 x_2 ... x_T)$, for $x_i \in \Sigma$, where Σ is a set of symbols, assume each x_i depends on the previous n variables in the sequence:

$$p(x_{1:T}) = \prod_{i=1}^{T} p(x_i | x_{(i-n+1):i-1})$$

- As n gets larger, computational complexity grows and probability of each n-gram occurring goes down, and smoothing is required.
- In the paper we covered last class, Teh used a Pitman-Yor process prior on the previous n words for Bayesian smoothing.

Non-Markov Model

- What if we we let n grow with the length of the data?
- Get a *non-Markov* model:

$$p(x_{1:T}) = \prod_{i=1}^{T} p(x_i | x_{1:i-1})$$

- Each symbol is conditioned on every previous symbol.
- How do we actually compute this?

The Sequence Memoizer Model

- For each symbol $s \in \Sigma$, and some context $\mathbf{u}$, create a latent variable $G_{\mathbf{u}} = [G_{\mathbf{u}}(s)]_{s \in \Sigma}$ (a probability vector). (I.e. $G_{\mathbf{u}}(s) = p(u_{T+1} = s | u_{1:T})$).
- Let $\mathcal{G} = \{G_{[\mathbf{s}]}\}_{\mathbf{s} \in \Sigma^*}$ be the (infinite) set of all such probability vectors for every possible sequence made from elements of Σ.
- So $p(\mathbf{x}_{1:T}, \mathcal{G}) = p(\mathcal{G}) \prod_{i=1}^{T} G_{\mathbf{x}_{1:i-1}}(x_i)$ for the particular sequences we observe. Notice that this is recursive.
- But what is $p(\mathcal{G})$?

The Sequence Memoizer Model

- Take $\mathcal{G}$ to be a Pitman-Yor process prior.
- They set c, the concentration parameters, to always be 0. Zach's paper covers the more general case.
- With this choice, we can model the power-law properties of language.
- Will also use some nice marginalization properties later.

The Sequence Memoizer Model

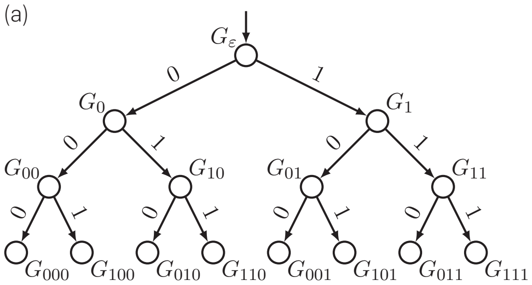
In particular, the Sequence Memoizer model gives a distribution over $\mathcal{G} = \{G_{\mathbf{u}}\}_{\mathbf{u} \in \Sigma^*}$ using hierarchical Pitman-Yor process:

$$\begin{aligned} G_{[]} | d_0, H &\sim \mathcal{PY}(d_0, 0, H) \\ G_{[\mathbf{u}]} | G_{[\sigma(\mathbf{u})]}, d_{[\mathbf{u}]} &\sim \mathcal{PY}(d_{|\mathbf{s}|}, 0, G_{[\sigma(\mathbf{u})]}) \ \forall \mathbf{u} \in \Sigma^+ \\ x_i | \mathbf{x}_{1:i-1} = \mathbf{u} &\sim G_{[\mathbf{u}]} \end{aligned}$$

where $\sigma(\mathbf{u})$ means the suffix of context $\mathbf{u}$ (e.g. if $\mathbf{u} = abcd$, $\sigma(\mathbf{u}) = bcd$).
Encodes prior knowledge that contexts sharing suffixes will be similar to each other, so later symbols in a context will be more important in prediction.

The Sequence Memoizer Model: Infinite

This hierarchy can be viewed as an infinite tree (a context tree), beginning at the empty root node, where each node has a branch for each $s \in \Sigma$. E.g. for $\Sigma = \{0, 1\}$:

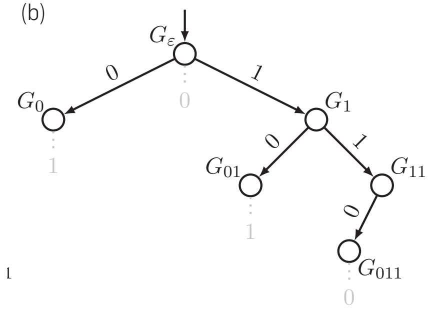


Source: Wood et al., "The Sequence Memoizer" CACM (2011).

The parent of node $\mathbf{u}$ is $\sigma(\mathbf{u})$, the *longest proper suffix* of that node. (E.g. 110 is the longest proper suffix of 0110).

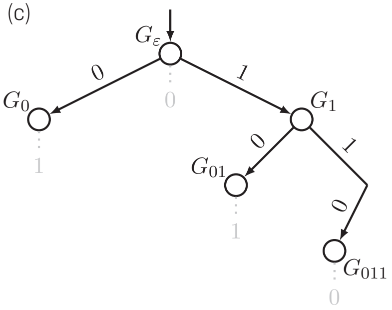
Prefix Trie: $O(T^2)$

- When given a particular sequence (e.g. $\mathbf{x} = 0110$), we can integrate all of the nodes in the context tree not associated with data in $\mathbf{x}$. The resulting tree looks like a suffix trie.
- Every prefix is a path in the tree.
- Requires $O(T^2)$ time and space to build this tree for a sequence of length T.
- Intuition: One-to-one correspondence between nodes of suffix trie and distinct substrings.



Prefix Tree: $O(T)$

- Obtained by compacting non branching, non leaf nodes.
- If we need those internal nodes, can recreate them.
- Better algorithm requires only $O(T)$ time and space to build! (At most $2T$ nodes.)



Source: Wood et al., "The Sequence Memoizer" CACM (2011).

Coagulation and Fragmentation

- Key concept: Compacting internal nodes of prefix trie $\Leftrightarrow$ marginalizing PYP.
- For certain parameter settings, chains of conditional PYP are closed under marginalization.
- Theorem: If $G_2|G_1 \sim \mathcal{PY}(d_1, 0, G_1)$ and $G_3|G_2 \sim \mathcal{PY}(d_2, 0, G_2)$, then $G_3|G_1 \sim \mathcal{PY}(d_1 d_2, 0, G_1)$ with G_2 marginalized out.
- Just multiply discount parameters along collapsed edge!
- Can also go backwards, to recreate G_2. Zach may go over this in more detail, shortly.

Inference Algorithm: Posterior

Intractable to do exact inference in this model, so they use a Gibbs sampler. Zach will probably cover this in his talk.

- Building the suffix tree for **x** gives the structure of a graphical model.
- Traverse that tree, collecting parameters for a hierarchical Pitman-Yor process.
- Instantiate a Chinese Restaurant Franchise representation of the HPYP.
- Use Gibbs sampling to simulate the posterior distribution conditioned on the observed sequence (as with any other CRF).

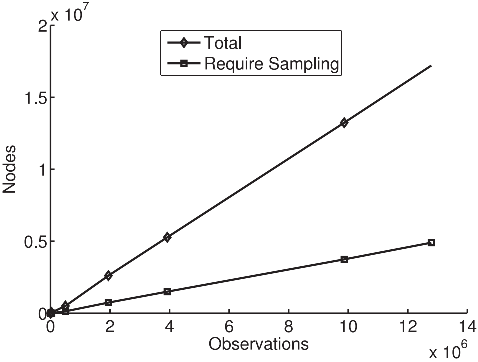
Inference Algorithm: Prediction

- Given some context $\mathbf{s}$ not in the training set, and some next symbol v, want to compute $p(v|\mathbf{s}, \mathbf{x})$.
- $p(v|\mathbf{s}, \mathbf{x}) = \mathbb{E}[G_{[\mathbf{s}]}(v)] = \mathbb{E}[G_{[\mathbf{s}']}(v)]$, where $\mathbf{s}'$ is the longest suffix of $\mathbf{s}$ in the prefix trie.
- If $\mathbf{s}'$ doesn't appear in the prefix tree, can use fragmentation to reinstate the corresponding restaurants into the model.
- $\mathbb{E}[G_{[\mathbf{s}]}(v)] = \mathbb{E}\left[\frac{N(\mathbf{s}v) - d_{|\mathbf{s}|} M(\mathbf{s}v) + \sum_{v' \in \Sigma} d_{|\mathbf{s}|} M(\mathbf{s}v') G_{\sigma(\mathbf{s})}(v)}{\sum_{v' \in \Sigma} N(\mathbf{s}v')}\right]$, where $\{N(\mathbf{s}'v'), M(\mathbf{s}'v')\}$ are random counts given some context s' and symbol v'.
- Use samples from posterior distribution to approximate this expectation.

Results

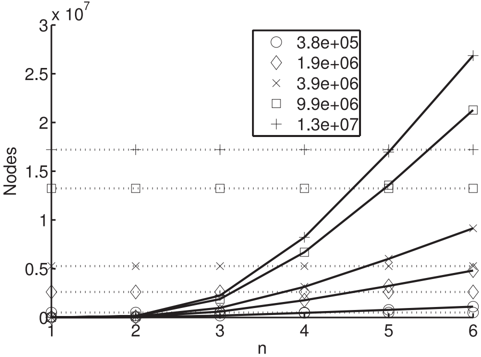
- On New York Times corpus and AP corpus
- Used CRF sampler with special Metropolis-Hastings updates for discount parameters (because collapsed nodes have products of discount parameters).
- Did really short burn-in (10 iterations) and collected 5 samples.

Results: Number of nodes in tree and number which require sampling



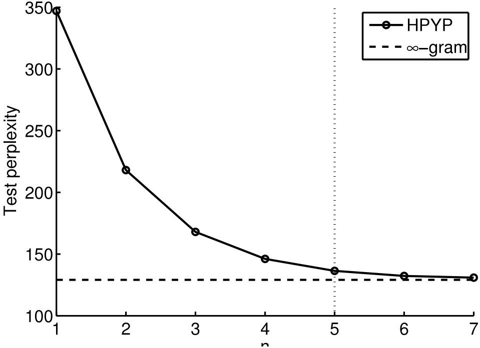
Shown as function of number of New York Times observations.
Grows linearly with corpus size. Leaf nodes don't require sampling.

Results: Nodes in prefix trees vs *n*-gram trie



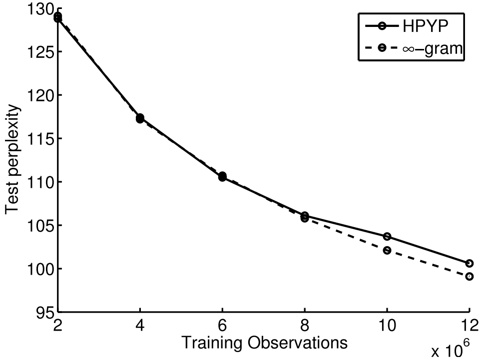
Horizontal lines are tree counts, curved lines are trie counts.
Note: as amount of data grows, tree has about same number of nodes as a 5-gram.

Results: Sequence Memoizer vs n-gram performance



The Sequence Memoizer always below HPYP. Af $n = 5$, HPYP begins to have more nodes (by previous figure).

Results



Using a 5-gram HPYP model. ∞-gram becomes better than 5-gram as dataset size increases.

Results

Source	Perplexity
(Mnih & Hinton, 2009)	112.1
(Bengio et al., 2003)	109.0
4-gram Modified Kneser-Ney (Teh, 2006)	102.4
4-gram HPYP (Teh, 2006)	101.9
∞-gram (Sequence Memoizer)	96.9

Very good perplexity results!

Application: Compression

- Gasthaus J, Wood F, Teh YW. "Lossless compression based on the Sequence Memoizer". DCC (2010).
- Used the predictive ability of the SM to very efficiently compress text.
- Developed an approximate incremental inference algorithm for the SM.

Application: Compression

		DEPLUMP		PPM		CTW
File	Size	1PF	UKN	PPM*	PPMZ	CTW
bib	111261	1.73	**1.72**	1.91	1.74	1.83
book1	768771	**2.17**	2.20	2.40	2.21	2.18
book2	610856	**1.83**	1.84	2.02	1.87	1.89
geo	102400	**4.40**	4.40	4.83	4.64	4.53
news	377109	**2.20**	2.20	2.42	2.24	2.35
obj1	21504	**3.64**	3.65	4.00	3.66	3.72
obj2	246814	2.21	**2.19**	2.43	2.23	2.40
paper1	53161	2.21	**2.20**	2.37	2.22	2.29
paper2	82199	**2.18**	2.18	2.36	2.21	2.23
pic	513216	0.77	0.82	0.85	**0.76**	0.80
progc	39611	2.23	**2.21**	2.40	2.25	2.33
progl	71646	1.44	**1.43**	1.67	1.46	1.65
progp	49379	1.44	**1.42**	1.62	1.47	1.68
trans	93695	1.21	**1.20**	1.45	1.23	1.44
avg.		**2.12**	2.12	2.34	2.16	2.24
w. avg.		**1.89**	1.91	2.09	1.93	1.99

In average bits/byte.

Final Note

- All of the code for the Sequence Memoizer is available online at `www.sequencememoizer.com`.
- There are C++ and Java implementations, and bindings to Python and R.