Efficient Computation of Expectations under Spanning Tree Distributions

September 8, 2021

Dependency Trees

- Classical representation of text
- Similar to phrase structure grammars
  - Phrase structure: Labels groups of words
  - Dependency: Labels relationship between pairs of words
- Useful if a language has free word order
- Spanning trees

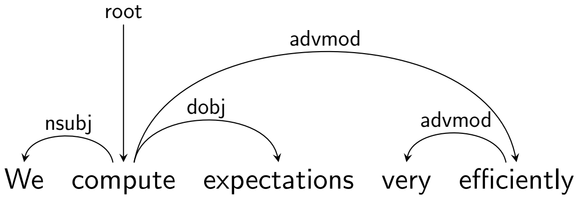

Figure: Example of a dependency tree

Efficient Computation of Expectations under Spanning Tree Distributions

- A framework for computing expectations (of decomposable functions) over spanning trees
  - Unifies previous algorithms
  - Additionally show large asymptotic and empirical speed improvements over particular past implementations
- Relies on connections between moments and derivatives
  - Uses automatic differentiation for easy to implement and efficient algorithms
- Fun demonstration of inference with automatic differentiation

Outline

Goal is to compute expectations over spanning trees

- Background: Distributions over (spanning) trees
- Method: Connecting expectations to derivatives
  - Use properties of **our** choice of tree distribution and decomposable functions
  - Stitch together into efficient algorithms with automatic differentiation
- Computational complexity

Distributions over Trees

Assuming fixed length sentence, with N nodes:

- Weighted edges

$$(i \xrightarrow{w_{ij}} j) \in \mathcal{E}$$

- Trees weights

$$w(d) := \prod_{(i \to j) \in d} w_{ij}$$

- Tree probability obtained via normalization

$$p(d) := \frac{w(d)}{\mathrm{Z}},$$

where

$$\mathrm{Z} := \sum_{d \in \mathcal{D}} w(d) = \sum_{d \in \mathcal{D}} \prod_{(i \to j) \in d} w_{ij}$$

- Computation of Z is tractable despite exponentially many trees in $\mathcal{D}$

Distributions over spanning trees

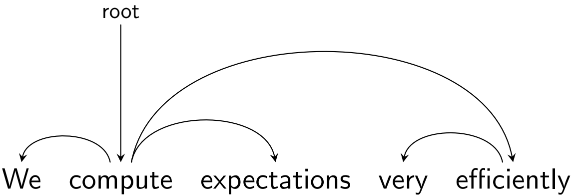

Figure: Example of a dependency tree

Matrix-Tree Theorem (MTT)

- Compute partition function Z using the graph Laplacian $L \in \mathbb{R}^{N \times N}$: $Z = |L|$.
  - Simple (undirected, unweights) graphs: $L = D - A$, where D is the degree matrix and A the adjacency
  - Weighted directed graphs:

$$L_{ij} := \begin{cases} \sum_{i' \in \mathcal{N} \setminus \{j\}} w_{i'j} & \textbf{if } i = j \\ -w_{ij} & \textbf{otherwise} \end{cases}$$

- Determinant can be computed in $O(N^3)$ time
- Flexibility in choice of L depending on example, care is needed
- Results are general for choice of Laplacian L

Expectations

- Main goal is to compute expectations / totals

$$\mathbb{E}_d\left[f(d)\right] := \sum_{d\in\mathcal{D}} p(d)f(d) = \frac{1}{Z}\sum_{d\in\mathcal{D}} w(d)f(d) = \frac{1}{Z}\overline{f}$$

- Need to consider every unique tree with nonzero mass unless $f : \mathcal{D} \to \mathbb{R}^F$ decomposes
- Consider two families of decomposable f
  - Additively decomposable: Shannon entropy, KL
  - Second-order additively decomposable: Gradient of entropy, covariance

Intuition for decompositions

- Recall: Tree distribution is a distribution over a binary vector of size $O(N^2)$
- Compute expectations by
  - Storing the relevant parts of trees and their marginal probabilities
  - Applying f to those parts
- Consider different levels of decomposability of f
  - Does not decompose: exponential num trees
  - Decomposes over pairs of edges: $O(N^4)$ evals
  - Decomposes over edges: $O(N^2)$ evals
- Combine $p(\text{part})f(\text{part})$ to compute expectation (unnormalized total using $p(\text{part}) = \frac{1}{Z}\tilde{w}_{\text{part}}$)

(Unnormalized) marginals

- Unary marginals (prob of one edge appearing in a tree)

$$p((i \to j) \in d) = \frac{1}{Z} \sum_{d \in \mathcal{D}_{ij}} w(d) = \frac{1}{Z} \widetilde{w_{ij}}$$

- Binary marginals (prob of two edges appearing together in a tree)

$$p((i \to j), (k \to l) \in d) = \frac{1}{Z} \sum_{d \in \mathcal{D}_{ij,kl}} w(d) = \frac{1}{Z} \widetilde{w_{ij,kl}}$$

- Want to show that
  - Can decompose totals into sums over marginals (enumerate parts of trees)
  - Can compute marginals using automatic differentation (cheap gradient principle)
  - Avoid storing full probability tables (requires low-level tricks, not the focus / see paper for details)

Additively decomposable functions

- Allows us to decompose into functions of edges and combine with (unnormalized) unary marginals

First order totals

- Rewrite unary marginal as gradient
- Rewrwite total as sum over unary parts

First order totals: Marginal

- A
- B

First order totals: Total

- A
- B

First order totals: Algorithm

- Rewrite unary marginal as gradient of partition function
- Rewrwite first order total as sum over unary parts

Second-order additively decomposable functions

- Allows us to decompose into functions of pairs of edges and combine with (unnormalized) binary marginals

Second-order totals

- Rewrite binary marginal as Hessian of partition function
- Rewrite grad of intermediate total as Hessian-vector product
- Rewrite second-order total as Jacobian-matrix product or Hessian-matrix product

Second-order totals: Marginal

- blah

Second-order totals: Grad of intermediate total

- asdf

Second-order totals: Total

- asdf

Refs I

[1] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[2] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 1723–1732, 2019.