# 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

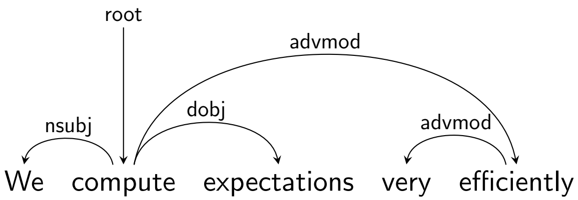


Figure: Example of a dependency tree

# 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 a general 'inference with automatic differentiation' recipe
  - Compute partition function
  - Use AD to compute expectations

# 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

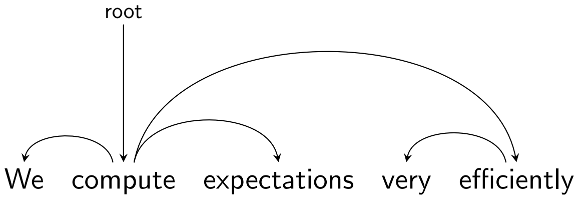


Figure: Example of a dependency tree

# 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 $p(\mathrm{part})f(\mathrm{part})$ to compute expectation (unnormalized total using $p(\mathrm{part}) = \frac{1}{Z}\tilde{w}_{\mathrm{part}}$)

# (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, simple implementation)
  - Avoid storing full probability tables (requires low-level tricks, not our focus / see paper for details)

# Additively decomposable functions

Allows us to decompose $r : \mathcal{D} \rightarrow \mathbb{R}^R$ into functions of edges and combine with (unnormalized) unary marginals (over those tree edges)

$$r(d) = \sum_{(i \rightarrow j) \in d} r_{ij}, \tag{1}$$

with $r_{ij} \in \mathbb{R}^R$

# Additively decomposable functions: Marginals

Marginals are expectations

$$p((i \rightarrow j) \in d) = \mathbb{E}_{p(d)}\left[1((i \rightarrow j) \in d)\right]_{ij} = [\sum_d p(d)r(d)]_{ij}$$

Set

$$r(d) = 1((i \rightarrow j) \in d)$$

Proof

$$1((i \rightarrow j) \in d) = \sum_{(k \rightarrow l) \in d} 1(kl = ij)$$

$$\Rightarrow r_{ij} = 1(kl = ij)$$

# Additively decomposable functions: Shannon Entropy

Shannon entropy is an expectation

$$\mathbb{E}_{p(d)}\left[-\log p(d)\right]$$

Set

$$r(d) = -\log p(d)$$

Proof

$$\begin{aligned} -\log p(d) &= -\log\left(\frac{1}{\mathrm{Z}} \prod_{(i \to j) \in d} w_{ij}\right) \\ &= \log \mathrm{Z} - \sum_{(i \to j) \in d} \log w_{ij} \\ \Rightarrow r_{ij} &= \frac{1}{N} \log \mathrm{Z} - \log w_{ij} \end{aligned}$$

# First order totals: Recipe

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

# First order totals: Marginal

For any edge $(i \to j)$,

$$\widetilde{w_{ij}} := \sum_{d \in \mathcal{D}_{ij}} w(d) = \frac{\partial \mathrm{Z}}{\partial w_{ij}} w_{ij}, \tag{2}$$

where $\mathcal{D}_{ij} := \{d \in \mathcal{D} \mid (i \to j) \in d\}$

- Significantly easier to implement via AD than manually

# First order totals: Marginal derivation

$$
\begin{aligned}
\widetilde{w_{ij}} &= \sum_{d \in \mathcal{D}_{ij}} w(d) && \text{(Defn)} \\
&= \sum_{d \in \mathcal{D}_{ij}} \prod_{(i' \to j') \in d} w_{i'j'} && \text{(Defn of } w(d)\text{)} \\
&= w_{ij} \sum_{d \in \mathcal{D}_{ij}} \prod_{\substack{(i' \to j') \in \\ d \setminus \{(i \to j)\}}} w_{i'j'} && \text{(Pull out } w_{ij}\text{)} \\
&= w_{ij} \frac{\partial}{\partial w_{ij}} \sum_{d \in \mathcal{D}} \prod_{(i' \to j') \in d} w_{i'j'} && \text{(Sum-prod deriv trick)} \\
&= \frac{\partial \mathrm{Z}}{\partial w_{ij}} w_{ij} && \text{(Defn of } Z\text{)}
\end{aligned}
$$

# First order totals: Total

For any additively decomposable function $r \colon \mathcal{D} \mapsto \mathbb{R}^R$,

$$\overline{r} = \sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} r_{ij} = \sum_{(i \to j) \in \mathcal{E}} \frac{\partial \mathrm{Z}}{\partial w_{ij}} w_{ij} r_{ij} \tag{3}$$

Proof.

$$\begin{aligned}
\overline{r} &= \sum_{d \in \mathcal{D}} w(d) r(d) && \text{(Defn)} \\
&= \sum_{d \in \mathcal{D}} w(d) \sum_{(i \to j) \in d} r_{ij} && \text{(Defn of } r(d)\text{)} \\
&= \sum_{d \in \mathcal{D}} \sum_{(i \to j) \in d} w(d) r_{ij} && \text{(Distributive prop)} \\
&= \sum_{(i \to j) \in \mathcal{E}} \sum_{d \in \mathcal{D}_{ij}} w(d) r_{ij} && \text{(Commutative prop)} \\
&= \sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} r_{ij} && \text{(Defn of } \widetilde{w_{ij}}\text{)}
\end{aligned}$$

# First order totals: Algorithm

**def** $\mathtt{T}_1\left(w\colon \mathcal{E} \mapsto \mathbb{R}, r\colon \mathcal{E} \mapsto \mathbb{R}^R\right)$ :

  ▷ *Compute first-order total; requires* $\mathcal{O}\left(N^3 R'\right)$ *time,* $\mathcal{O}\left(N^2 + R\right)$ *space.*

  Compute all $\widetilde{w_{ij}}$ via AD in $\mathcal{O}\left(N^3\right)$[1]

  $\overline{r} \leftarrow \sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} r_{ij}$  ▷ $\mathcal{O}\left(N^2 R'\right)$

  **return** $\overline{r}$

[1] Constant factor of $|L|$ runtime with AD

# Second-order additively decomposable functions

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

$$t(d) = r(d)s(d)^\top \tag{4}$$

- Outer product of two additively decomposable functions, $r\colon \mathcal{D} \mapsto \mathbb{R}^R$ and $s\colon \mathcal{D} \mapsto \mathbb{R}^S$
- $t(d) \in \mathbb{R}^{R\times S}$ is generally a matrix.

# Second-order additively decomposable functions: Binary marginals

Binary marginals are expectations

$$p((i \to j),(k \to l) \in d) = \mathbb{E}_{p(d)}\left[1((i \to j),(k \to l) \in d)\right]_{ij,kl}$$
$$= \left[\sum_d p(d)r(d)s(d)^\top\right]_{ij,kl}$$

Set

$$r(d) = 1((i \to j) \in d), \qquad s(d) = 1((k \to l) \in d)$$

which we know are additively decomposable

## Second-order totals: Recipe

- 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

Unnormalized binary marginals

$$\widetilde{w_{ij,kl}} := \sum_{d \in \mathcal{D}_{ij,kl}} w(d) = \frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\, \partial w_{kl}} w_{ij} w_{kl} \tag{5}$$

where $\mathcal{D}_{ij,kl} := \{d \in \mathcal{D} \mid (i \to j) \in d, (k \to l) \in d\}$

# Second-order totals: Marginal

Same tricks as first-order marginals

$$\begin{aligned}
\widetilde{w_{ij,kl}} &= \sum_{d \in \mathcal{D}_{ij,kl}} w(d) \\
&= \sum_{d \in \mathcal{D}_{ij,kl}} \prod_{(k' \to l') \in d} w_{k'l'} \\
&= w_{ij} w_{kl} \frac{\partial^2}{\partial w_{ij} \partial w_{kl}} \sum_{d \in \mathcal{D}} \prod_{(i' \to j') \in d} w_{i'j'} \\
&= \frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\, \partial w_{kl}} w_{ij} w_{kl}
\end{aligned}$$

# Second-order totals: Grad of intermediate total

Write $\nabla \bar{r}$ in terms of Hessian $\nabla^2 Z$, as a step towards writing $\bar{t}$ in terms of $\nabla^2 Z$

$$
\begin{aligned}
& w_{ij} \frac{\partial \bar{r}}{\partial w_{ij}} \\
&= w_{ij} \frac{\partial}{\partial w_{ij}} \left( \sum_{(k \to l) \in \mathcal{E}} \frac{\partial \mathrm{Z}}{\partial w_{kl}} w_{kl} r_{kl} \right) \\
&= w_{ij} \frac{\partial \mathrm{Z}}{\partial w_{ij}} r_{ij} + w_{ij} \sum_{(k \to l) \in \mathcal{E}} \frac{\partial^2 \mathrm{Z}}{\partial w_{ij} \partial w_{kl}} w_{kl} r_{kl} \\
&= \widetilde{w_{ij}} r_{ij} + \sum_{(k \to l) \in \mathcal{E}} \widetilde{w_{ij,kl}} r_{kl}
\end{aligned}
$$

# Second-order totals: Total

We can both

- Write total $\bar{t}$ in terms of Jacobian $\frac{\partial \bar{r}}{\partial w_{ij}}$ or gradients $\nabla \bar{r}_n$
  - Run backward for each dimension or $\bar{r}$: $O(RN^3 + R^2N^2)$
- Write total $\bar{t}$ in terms of Hessian $\nabla^2 Z$
  - Hessian is large $O(N^4)$ entries
  - This takes $O(N^4)$ time, but can be optimized to $O(N^2 + RS)$ space

# Second-order totals: Total

$$
\begin{aligned}
\bar{t} &= \sum_{d \in \mathcal{D}} w(d) r(d) s(d)^\top \\
&= \sum_{d \in \mathcal{D}} w(d) r(d) \sum_{(i \to j) \in d} s_{ij}{}^\top \\
&= \sum_{d \in \mathcal{D}} \sum_{(i \to j) \in d} w(d) r(d) s_{ij}{}^\top \\
&= \sum_{(i \to j) \in \mathcal{E}} \sum_{d \in \mathcal{D}_{ij}} w(d) r(d) s_{ij}{}^\top \\
&= \sum_{(i \to j) \in \mathcal{E}} w_{ij} \frac{\partial}{\partial w_{ij}} \left( \sum_{d \in \mathcal{D}} w(d) r(d) \right) s_{ij}{}^\top \\
&= \sum_{(i \to j) \in \mathcal{E}} w_{ij} \frac{\partial \bar{r}}{\partial w_{ij}} s_{ij}{}^\top
\end{aligned}
$$

# Second-order totals: Jacobian version

**def** $\mathtt{T}_2^{\mathtt{v}}\left(w\colon \mathcal{E} \mapsto \mathbb{R}, r\colon \mathcal{E} \mapsto \mathbb{R}^R, s\colon \mathcal{E} \mapsto \mathbb{R}^S\right)$ :

  ▷ *Compute second-order total with gradient-vector products; requires* $\mathcal{O}(R(N^3+N^2R'+N^2S'))$ *time,* $\mathcal{O}\left(N^2R+R\,S\right)$ *space.*

  **for** $n = 1 \ldots R$ :  ▷ $\mathcal{O}\left(R(N^3+N^2R')\right)$

    Compute $\nabla \overline{r}_n$ using reverse-mode AD on $[\mathtt{T}_1(w, r)]_n$

  ▷ *Requires* $\mathcal{O}\left(N^2RS'\right)$

  **return** $\sum_{(i \to j) \in \mathcal{E}} \frac{\partial \overline{r}}{\partial w_{ij}} w_{ij} {s_{ij}}^\top$

- Recommended approach

# Second-order totals: Hessian version

**def** $\mathrm{T}_2^{\mathrm{h}}\left(w\colon \mathcal{E} \mapsto \mathbb{R}, r\colon \mathcal{E} \mapsto \mathbb{R}^R, s\colon \mathcal{E} \mapsto \mathbb{R}^S\right)$ :

$\quad \triangleright$ *Compute second-order total by materializing Hessian; requires $\mathcal{O}\left(N^4 R' S'\right)$ time, $\mathcal{O}\left(N^2 + R\, S\right)$ space.*

$\quad$ Compute all $\widetilde{w_{ij}}$ via $\nabla Z$

$\quad$ Compute all $\widetilde{w_{ij,kl}}$ via $\nabla^2 Z$ (can compute this efficiently)

$\quad \triangleright$ *Requires $\mathcal{O}\left(N^4 R' S'\right)$*

$\quad$ **return** $\sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} r_{ij} {s_{ij}}^\top + \sum_{(k \to l) \in \mathcal{E}} \widetilde{w_{ij,kl}} r_{ij} {s_{kl}}^\top$

- Better than previous algo at dealing with sparsity, i.e. if $R$ and $S$ are really big and sparse
- Can come up with a 3rd algo that has the best of both worlds by computing the second term involving the Hessian-matrix product more efficiently

# 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.