Efficient Computation of Expectations under Spanning Tree Distributions

Ran Zmigrod [✨,🌲] **Tim Vieira** [✨,🌴] **Ryan Cotterell** [🌲,🌳]

[🌲]University of Cambridge, United Kingdom [🌴]Johns Hopkins University, United States

[🌳]ETH Zürich, United Kingdom

rz279@cam.ac.uk tim.f.vieira@gmail.com

ryan.cotterell@inf.ethz.ch

Abstract

We give a general framework for inference in spanning tree models. We propose unified algorithms for the important cases of first-order expectations and second-order expectations in edge-factored, non-projective spanning-tree models. Our algorithms exploit a fundamental connection between gradients and expectations, which allows us to derive efficient algorithms. These algorithms are easy to implement with or without automatic differentiation software. We motivate the development of our framework with several *cautionary tales* of previous research, which has developed numerous inefficient algorithms for computing expectations and their gradients. We demonstrate how our framework efficiently computes several quantities with known algorithms, including the expected attachment score, entropy, and generalized expectation criteria. As a bonus, we give algorithms for quantities that are missing in the literature, including the KL divergence. In all cases, our approach matches the efficiency of existing algorithms and, in several cases, reduces the runtime complexity by a factor of the sentence length. We validate the implementation of our framework through runtime experiments. We find our algorithms are up to 15 and 9 times faster than previous algorithms for computing the Shannon entropy and the gradient of the generalized expectation objective, respectively.

1 Introduction

Dependency trees are a fundamental combinatorial structure in natural language processing. It follows that probability models over dependency trees are an important object of study. In terms of graph theory, one can view a (non-projective) dependency tree as an arborescence (commonly known as a spanning tree) of a graph. To build a dependency parser, we define a graph where the nodes are the tokens of the sentence, and the edges are possible dependency relations between the tokens. The edge weights are defined by a model, which is learned from data. In this paper, we focus on edge-factored models where the probability of a dependency tree is proportional to the product the weights of its edges. As there are exponentially many trees in the length of the sentence, we require clever algorithms for finding the normalization constant. Fortunately, the normalization constant for edge-factored models is efficient to compute via to the celebrated matrix–tree theorem.

The matrix–tree theorem (Kirchhoff, 1847)—more specifically, its counterpart for directed graphs (Tutte, 1984)—appeared before the NLP community in an onslaught of contemporaneous papers (Koo et al., 2007; McDonald and Satta, 2007; Smith and Smith, 2007) that leverage the classic result to efficiently compute the normalization constant of a distribution over trees. The result is still used in more recent work (Ma and Hovy, 2017; Liu and Lapata, 2018). We build upon this tradition through a framework for computing expectations of a rich family of functions under a distribution over trees. Expectations appear in all aspects of the probabilistic modeling process: training, model validation, and prediction. Therefore, developing such a framework is key to accelerating progress in probabilistic modeling of trees.

Our framework is motivated by the lack of a unified approach for computing expectations over spanning trees in the literature. We believe this gap has resulted in the publication of numerous inefficient algorithms. We motivate the importance of developing such a framework by highlighting the following *cautionary tales*.

- McDonald and Satta (2007) proposed an inefficient $\mathcal{O}(N^5)$ algorithm for computing

[✨]Equal contribution.

Transactions of the Association for Computational Linguistics, vol. 9, pp. 675–690, 2021. https://doi.org/10.1162/tacl_a_00391
Action Editor: Dan Gildea. Submission batch: 5/2020; Revision batch: 3/2021; Published 7/2021.

feature expectations, which was much slower than the $\mathcal{O}(N^3)$ algorithm obtained by Koo et al. (2007) and Smith and Smith (2007). The authors subsequently revised their paper.

- Smith and Eisner (2007) proposed an $\mathcal{O}(N^4)$ algorithm for computing entropy. Later, Martins et al. (2010) gave an $\mathcal{O}(N^3)$ method for entropy, but not its gradient. Our framework recovers Martins et al.'s (2010) algorithm, and additionally provides the gradient of entropy in $\mathcal{O}(N^3)$.

- Druck et al. (2009) proposed an $\mathcal{O}(N^5)$ algorithm for evaluating the gradient of the generalized expectation (GE) criterion (McCallum et al., 2007). The runtime bottleneck of their approach is the evaluation of a covariance matrix, which Druck and Smith (2009) later improved to $\mathcal{O}(N^4)$. We show that the gradient of the GE criterion can be evaluated in $\mathcal{O}(N^3)$.

We summarize our main results below:

- **Unified Framework**: We develop an algorithmic framework for calculating expectations over spanning arborescences. We give precise mathematical assumptions on the types of functions that are supported. We provide efficient algorithms that piggyback on automatic differentiation techniques, as our framework is rooted in a deep connection between expectations and gradients (Darwiche, 2003; Li and Eisner, 2009).

- **Improvements to existing approaches**: We give asymptotically faster algorithms where several prior algorithms were known.

- **Efficient algorithms for new quantities**: We demonstrate how our framework calculates several new quantities, such as the Kullback–Leibler divergence, which (to our knowledge) had no prior algorithm in the literature.

- **Practicality**: We present practical speed-ups in the calculation of entropy compared to Smith and Eisner (2007). We observe speed-ups in the range of 4.1 and 15.1 in five languages depending on the typical sentence length. We also demonstrate a 9 times speed-up for evaluating the gradient of the GE objective compared to Druck and Smith (2009).

- **Simplicity**: Our algorithms are simple to implement—requiring only a few lines of PyTorch code (Paszke et al., 2019). We have released a reference implementation at the following URL: `https://github.com/rycolab/tree_expectations`.

2 Distributions over Trees

We consider the distribution over trees in weighted directed graphs with a designated root node. A (rooted, weighted, and directed) **graph** is given by $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \rho)$. $\mathcal{N} = \{1, \ldots, N\} \cup \{\rho\}$ is a set of $N+1$ nodes where ρ is a designated root node. $\mathcal{E}$ is a set of weighted edges where each edge $(i \xrightarrow{w_{ij}} j) \in \mathcal{E}$ is a pair of *distinct* nodes such that the source node $i \in \mathcal{N}$ points to a destination node $j \in \mathcal{N}$ with an edge weight $w_{ij} \in \mathbb{R}$. We assume—without loss of generality—that the root node ρ has no incoming edges. Furthermore, we assume only one edge can exist between two nodes. We consider the multi-graph case in §2.2.

In natural language processing applications, these weights are typically parametric functions, such as log-linear models (McDonald et al., 2005b) or neural networks (Dozat and Manning, 2017; Ma and Hovy, 2017), which are learned from data.

A **tree**[1] d of a graph $\mathcal{G}$ is a set of N edges such that all non-root nodes j have exactly one incoming edge and the root node ρ has at least one outgoing edge. Furthermore, a tree does not contain any cycles. We denote the set of all trees in a graph by $\mathcal{D}$ and assume that $|\mathcal{D}| > 0$ (this is not necessarily true for all graphs).

The **weight of a tree** $d \in \mathcal{D}$ is defined as:

$$w(d) \stackrel{\text{def}}{=} \prod_{(i \to j) \in d} w_{ij} \tag{1}$$

Normalizing the weight of each tree yields a **probability distribution**:

$$p(d) \stackrel{\text{def}}{=} \frac{w(d)}{\mathrm{Z}} \tag{2}$$

where the **normalization constant** is defined as

$$\mathrm{Z} \stackrel{\text{def}}{=} \sum_{d \in \mathcal{D}} w(d) = \sum_{d \in \mathcal{D}} \prod_{(i \to j) \in d} w_{ij} \tag{3}$$

[1]The more precise graph-theoretic term is *arborescence*.

Of course, for (2) to be a *proper* distribution, we require $w_{ij} \geq 0$ for all $(i \rightarrow j) \in \mathcal{E}$, and $\mathrm{Z} > 0$.

2.1 The Matrix–Tree Theorem

The normalization constant Z involves a sum over $\mathcal{D}$, which can grow exponentially large with N. Fortunately, there is sufficient structure in the computation of Z that it can be evaluated in $\mathcal{O}(N^3)$ time. The Matrix–Tree Theorem (MTT) (Tutte, 1984; Kirchhoff, 1847) establishes a connection between Z and the determinant of the **Laplacian matrix**, $\mathbf{L} \in \mathbb{R}^{N \times N}$. For all $i, j \in \mathcal{N} \setminus \{\rho\}$,

$$\mathrm{L}_{ij} \stackrel{\text{def}}{=} \begin{cases} \sum_{i' \in \mathcal{N} \setminus \{j\}} w_{i'j} & \textbf{if } i = j \\ -w_{ij} & \textbf{otherwise} \end{cases} \tag{4}$$

Theorem 1 (Matrix–Tree Theorem; Tutte (1984, p. 140))**.** *For any graph,*

$$\mathrm{Z} = |\mathbf{L}| \tag{5}$$

Furthermore, the normalization constant can be computed in $\mathcal{O}(N^3)$ time.[2]

2.2 Dependency parsing and the Laplacian Zoo

Graph-based dependency parsing can be encoded as follows. For each sentence of length N, we create a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \rho)$ where each non-root node represents a token of the sentence, and ρ represents a special root symbol of the sentence. Each edge $(i \rightarrow j)$ in the graph represents a *possible* dependency relation between head word i and modifier word j. Fig. 1 gives an example dependency tree. In the remainder of this section, we give several variations on the Laplacian matrix that encode different sets of valid trees.[3]

In many cases of dependency parsing, we want ρ to have exactly one outgoing edge. This is motivated by linguistic theory, where the root of a sentence should be a token in the sentence rather than a special root symbol (Tesnière, 1959). There are exceptions to this, such as parsing Twitter (Kong et al., 2014) and parsing specific languages (e.g., The Prague Treebank [Bejček et al., 2013]). We call these **multi-root trees**[4] and these are represented by the set $\mathcal{D}$, as described earlier. Therefore, the normalization constant over all multi-root trees can be computed by a direct application of Theorem 1.

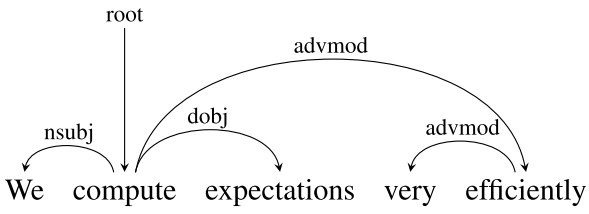

Figure 1: Example of a dependency tree.

However, in most dependency parsing corpora, only one edge may emanate from the root (Nivre et al., 2018; Zmigrod et al., 2020). Thus, we consider the set of **single-rooted trees**, denoted $\mathcal{D}^{(1)}$. Koo et al. (2007) adapt Theorem 1 to efficiently compute Z for the set $\mathcal{D}^{(1)}$ with the **root-weighted Laplacian**,[5] $\widehat{\mathbf{L}} \in \mathbb{R}^{N \times N}$

$$\widehat{\mathrm{L}}_{ij} = \begin{cases} w_{\rho j} & \textbf{if } i = 1 \\ \sum_{i' \in \mathcal{N} \setminus \{\rho, j\}} w_{i'j} & \textbf{if } i = j \\ -w_{ij} & \textbf{otherwise} \end{cases} \tag{6}$$

Proposition 1. *For any graph, the normalization constant over all single-rooted trees is given by the determinant of the root-weighted Laplacian (Koo et al., 2007, Prop. 1)*

$$\mathrm{Z} = |\widehat{\mathbf{L}}| \tag{7}$$

Furthermore, the normalization constant for single-rooted trees can be computed in $\mathcal{O}(N^3)$ time.

Labeled Trees. To encode *labeled* dependency relations in our set of trees, we simply augment edges with labels—resulting in a **multi-graph** in which multiple edges may exist between pairs of nodes. Now, edges take the form $(i \xrightarrow{y/\, w_{ijy}} j)$ where i and j are the source and destination nodes as before, $y \in \mathcal{Y}$ is the label, and w_{ijy} is their weight.

[2] For simplicity, we assume that the runtime of matrix determinants is $\mathcal{O}(N^3)$. However, we would be remiss if we did not mention that algorithms exist to compute the determinant more efficiently (Dumas and Pan, 2016).

[3] The reader may want to skip this section on their first reading.

[4] We follow the conventions of Koo et al. (2007) and say ''single-root'' and ''multi-root'' when we *technically* mean the number of outgoing edges from the root ρ, and *not* the number of root nodes in a tree, which is always one.

[5] The choice to replace row 1 by the root edges is done by convention, we can replace *any* row in the construction of $\widehat{\mathbf{L}}$.

Proposition 2. *For any multi-graph, the normalization constant for multi-root or single-rooted trees can be calculated using Theorem 1 or Proposition 1 (respectively) with the edge weights,*

$$w_{ij} = \sum_{y \in \mathcal{Y}} w_{ijy} \tag{8}$$

Furthermore, the normalization constant can be computed in $\mathcal{O}(N^3 + |\mathcal{Y}|N^2)$ *time.*[6]

Summary. We give common settings in which the MTT can be adapted to efficiently compute Z for different sets of trees. The choice is dependent upon the task of interest, and one must be careful to choose the correct Laplacian configuration. The results we present in this paper are modular in the specific choice of Laplacian. For the remainder of this paper, we assume the unlabeled tree setting and will refer to the set of trees as simply $\mathcal{D}$ and our choice of Laplacian as $\mathbf{L}$.

3 Expectations

In this section, we characterize the family of expectations that our framework supports. Our framework is an extension of Li and Eisner (2009) to distributions over spanning trees. In contrast, their framework considers expectations over distributions that can be factored as B-hypergraphs (Gallo et al., 1993). Our distributions over trees cannot be cast as polynomial-size B-hypergraphs. Another important distinction between our framework and that of Li and Eisner (2009) is that we do not use the semiring abstraction as it is algebraically too weak to compute the determinant efficiently.[7]

The **expected value** of a function $f : \mathcal{D} \mapsto \mathbb{R}^F$ is defined as follows

$$\mathbb{E}_d[f(d)] \stackrel{\text{def}}{=} \sum_{d \in \mathcal{D}} p(d) f(d) \tag{9}$$

Without any assumptions on f, computing (9) is intractable.[8] In the remainder of this section, we will characterize a class of functions f whose expectations can be efficiently computed.

The first type of functions we consider are functions that are **additively decomposable** along the edges of the tree. Formally, a function $r : \mathcal{D} \mapsto \mathbb{R}^R$ is additively decomposable if it can be written as

$$r(d) = \sum_{(i \to j) \in d} r_{ij} \tag{10}$$

where we abuse notation slightly by for any function $r : \mathcal{D} \mapsto \mathbb{R}^R$, we consider the edge function r_{ij} as a vector of edge values. An example of an additively decomposable function is $r(d) = -\log p(d)$ whose expectation gives the Shannon entropy.[9] Other first-order expectations include the expected attachment score and the Kullback–Leibler divergence. We demonstrate how to compute these in our framework in and §6.1 and §6.3, respectively.

The second type of functions we consider are functions that are **second-order additively decomposable** along the edges of the tree. Formally, a function r: $\mathcal{D} \mapsto \mathbb{R}^R$ is second-order additively decomposable if it can be written as the outer product of two additively decomposable functions, $r : \mathcal{D} \mapsto \mathbb{R}^R$ and $s : \mathcal{D} \mapsto \mathbb{R}^S$

$$t(d) = r(d)s(d)^\top \tag{11}$$

Thus, $t(d) \in \mathbb{R}^{R \times S}$ is generally a matrix.

An example of such a function is the gradient of entropy (see §6.2) or the GE objective (McCallum et al., 2007) (see §6.4 with respect to the edge weights. Another example of a second-order additively decomposable function is the

[6]The algorithms given in later sections will not provide full details for the labeled case due to space constraints, but we assure the reader that our algorithms can be straightforwardly generalized to the labeled setting.

[7]In fact, Jerrum and Snir (1982) proved that the partition function for spanning trees requires an exponential number of additions and multiplications in the semiring model of computation (i.e., assuming that subtraction is not allowed). Interestingly, division is not required, but algorithms for division-free determinant computation run in $\mathcal{O}(N^4)$ (Kaltofen, 1992). An excellent overview of *the power of subtraction* in the context of dynamic programming is given in Miklós (2019, Ch. 3). It would appear as if commutative rings would make a good level of abstraction as they admit efficient determinant computation. Interestingly, this means that we cannot use the MTT in the max-product semiring to (efficiently) find the maximum weight tree because max does not have an inverse. Fortunately, there exist $\mathcal{O}(N^2)$ algorithms to find the maximum weight tree for both the single-root and multi-root settings (Zmigrod et al., 2020; Gabow and Tarjan, 1984).

[8]Of course, one could use sampling methods, such as Monte Carlo, to approximate (9). Sampling methods may be efficient if the variance of f under p is not too large.

[9]Proof: $-\log p(d) = -\log(\frac{1}{\mathrm{Z}} \prod_{(i \to j) \in d} w_{ij})$
$= \log \mathrm{Z} - \sum_{(i \to j) \in d} \log w_{ij}$.
$\Rightarrow r_{ij} = \frac{1}{N} \log \mathrm{Z} - \log w_{ij}$.

covariance matrix. Given two feature functions $r\colon \mathcal{D} \mapsto \mathbb{R}^R$ and $s\colon \mathcal{D} \mapsto \mathbb{R}^S$, their covariance matrix is $\mathbb{E}_d\big[r(d)s(d)^\top\big] - \mathbb{E}_d[r(d)]\mathbb{E}_d[s(d)]^\top$. Thus, it is second-order additively decomposable function as long as $r(d)$ and $s(d)$ are additively decomposable.

One family of functions which can be computed efficiently but we will not explore here are those who are **multiplicatively decomposable** over the edges. A function $q\colon \mathcal{D} \mapsto \mathbb{R}^Q$ is multiplicatively decomposable if it can be written as

$$q(d) = \prod_{(i \to j) \in d} q_{ij} \tag{12}$$

where the product of q_{ij} is an element-wise vector product. These functions form a family that we will call zero[th]-order expectations and can be computed with a constant number of calls to MTT (usually two or three). Examples of these include the Rényi entropy and ℓ_p-norms.[10]

4 Connecting Gradients and Expectations

In this section, we build upon a fundamental connection between gradients and expectations (Darwiche, 2003; Li and Eisner, 2009). This connection allows us to build on work in automatic differentiation to obtain efficient gradient algorithms. While the propositions in this section are inspired from past work, we believe that the presentation and proofs of these propositions have previously not been clearly presented.[11] We find it convenient to work with unnormalized expectations, or totals (for short). We denote the **total** of a function f as $\overline{f} \stackrel{\text{def}}{=} \sum_{d \in \mathcal{D}} w(d) f(d)$. We recover the expectation with $\mathbb{E}_p[f] = \overline{f}/\mathrm{Z}$. We note that totals (on their own) may be of interest in some applications (Vieira and Eisner, 2017, Section 5.3).

The FIrst-Order Case. Specifically, the partial derivative $\frac{\partial \mathrm{Z}}{\partial w_{ij}}$ is useful for determining the **total weight** of trees which include the edge $(i \to j)$,

$$\widetilde{w_{ij}} \stackrel{\text{def}}{=} \sum_{d \in \mathcal{D}_{ij}} w(d) \tag{13}$$

where $\mathcal{D}_{ij} \stackrel{\text{def}}{=} \{d \in \mathcal{D} \mid (i \to j \in d)\}$. Furthermore, $p((i \to j) \in d) = \widetilde{w_{ij}}/\mathrm{Z} = \frac{w_{ij}}{\mathrm{Z}} \frac{\partial \mathrm{Z}}{\partial w_{ij}}$.[12]

Proposition 3. *For any edge $i \to j$,*

$$\widetilde{w_{ij}} = \frac{\partial \mathrm{Z}}{\partial w_{ij}} w_{ij} \tag{14}$$

Proof.

$$\begin{aligned}
\widetilde{w_{ij}} &= \sum_{d \in \mathcal{D}_{ij}} w(d) \\
&= \sum_{d \in \mathcal{D}_{ij}} \prod_{(i' \to j') \in d} w_{i'j'} \\
&= w_{ij} \sum_{d \in \mathcal{D}_{ij}} \prod_{\substack{(i' \to j') \in \\ d \setminus \{i \to j\}}} w_{i'j'} \\
&= w_{ij} \frac{\partial}{\partial w_{ij}} \sum_{d \in \mathcal{D}} \prod_{(i' \to j') \in d} w_{i'j'} \\
&= w_{ij} \frac{\partial}{\partial w_{ij}} \sum_{d \in \mathcal{D}} \prod_{(i' \to j') \in d} w_{i'j'} \\
&= \frac{\partial \mathrm{Z}}{\partial w_{ij}} w_{ij}
\end{aligned}$$

□

Proposition‘4 will establish a connection between the unnormalized expectation $\overline{r}$ and $\nabla \mathrm{Z}$.

Proposition 4. *For any additively decomposable function $r\colon \mathcal{D} \mapsto \mathbb{R}^R$, the total $\overline{r}$ can be computed using a gradient–vector product*

$$\overline{r} = \sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} r_{ij} \tag{15}$$

[10] The ℓ_k norm of the distribution p often denoted as $\|p\|_k \stackrel{\text{def}}{=} \left(\sum_{d \in \mathcal{D}} p(d)^k\right)^{1/k}$ for $k \geq 0$. It is computable from a zero[th]-order expectation because it can be written as $(\frac{\mathrm{Z}^{(k)}}{\mathrm{Z}^k})^{1/k}$ where $\mathrm{Z}^{(k)} = \sum_{d \in \mathcal{D}} w(d)^k = \sum_{(i \to j) \in d} w_{ij}^k$, which is clearly a zero[th]-order expectation. Similarly, the Rényi entropy of order $\alpha \geq 0$ with $\alpha \neq 1$ is $\mathrm{H}_\alpha(p) \stackrel{\text{def}}{=} \frac{1}{1-\alpha} \log\left(\sum_{d \in \mathcal{D}} p(d)^\alpha\right) = \frac{1}{1-\alpha} \log\left(\frac{\mathrm{Z}^{(\alpha)}}{\mathrm{Z}^\alpha}\right)$.

[11] Li and Eisner (2009, Section 5.1) provide a similar derivation to Proposition 3 and Proposition 4 for hypergraphs.

[12] Some authors (e.g., Wainwright and Jordan, 2008) prefer to work with an exponentiated representation $w_{ij} = \exp(\theta_{ij})$ so that $\nabla_{\theta_{ij}} \log \mathrm{Z} = p((i \to j) \in d)$. This avoids an explicit division by Z, and multiplication by w_{ij} as these operations happens by virtue of the chain rule.

Proof.

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

□

The Second-Order Case. We can similarly use $\frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\,\partial w_{kl}}$ to determine the total weight of trees which include both $(i \to j)$ and $(k \to l)$ with $(i \to j) \neq (k \to l)$[13]

$$\widetilde{w_{ij,\,kl}} \stackrel{\text{def}}{=} \sum_{d\in\mathcal{D}_{ij,\,kl}} w(d) \tag{16}$$

where $\mathcal{D}_{ij} \stackrel{\text{def}}{=} \{d \in \mathcal{D} \mid (i \to j) \in d, (k \to l) \in d\}$. Furthermore, $\frac{\widetilde{w_{ij,\,kl}}}{\mathrm{Z}} = p(i \to j \in d, (k \to l) \in d)$.

Proposition 5. *For any pair of edges $i \to j$ and $(k \to l)$ such that $i \to j \neq (k \to l)$,*

$$\widetilde{w_{ij,kl}} = \frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\,\partial w_{kl}} w_{ij} w_{kl} \tag{17}$$

Proof.

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

□

Proposition 6 will relate $\nabla^2 \mathrm{Z}$ to $\nabla \overline{r}$. This will be used in Proposition 7 to establish a connection between the total $\bar{t}$ and $\nabla^2 \mathrm{Z}$, and additionally establishes a connection between $\bar{t}$ and $\nabla \overline{r}$.

[13] As each edge can only appear once in a tree, $\widetilde{w_{ij,\,ij}} = 0$.

Proposition 6. *For any additively decomposable function $r : \mathcal{D} \mapsto \mathbb{R}^R$ that does not depend on w,*[14] *and edge $i \to j \in \mathcal{E}$,*

$$w_{ij}\frac{\partial \overline{r}}{\partial w_{ij}} = \widetilde{w_{ij}} r_{ij} + \sum_{(k\to l)\in\mathcal{E}} \widetilde{w_{ij,\,kl}} r_{kl} \tag{18}$$

Proof.

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

□

Proposition 7. *For any second-order additively decomposable function $t\colon \mathcal{D} \mapsto \mathbb{R}^{R\times S}$, which is expressed as the outer product of additively decomposable functions, $r : \mathcal{D} \mapsto \mathbb{R}^R$ and $s\colon \mathcal{D} \mapsto \mathbb{R}^S$, $t(d) = r(d)s(d)^\top$, where r does not depend on w, the total $\bar{t}$ can be computed using a Jacobian–matrix product*

$$\bar{t} = \sum_{(i\to j)\in\mathcal{E}} \frac{\partial \overline{r}}{\partial w_{ij}} w_{ij} {s_{ij}}^\top \tag{19}$$

or a Hessian–matrix product

$$\bar{t} = \sum_{(i\to j)\in\mathcal{E}} \widetilde{w_{ij}} r_{ij} {s_{ij}}^\top + \sum_{(k\to l)\in\mathcal{E}} \widetilde{w_{ij,\,kl}} r_{ij} {s_{kl}}^\top \tag{20}$$

[14] More precisely, $\frac{\partial r(d)}{\partial w_{ij}} = \mathbf{0}$ for all $d \in \mathcal{D}$ and $i \to j \in \mathcal{E}$.

Proof. We first prove (19)

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

Then (20) immediately follows by substituting (18) into (19) and expanding the summation. □

Remark. There is a simple recipe to compute $\nabla \overline{r}_n$ for each $n = 1, \ldots, R$. First, some notation; let $\overrightarrow{\mathbf{1}_{ij}}$ be a vector over $\mathcal{E}$ with a 1 in dimension $(i \to j)$, and zeros elsewhere. By plugging $[r_{ij}]_n$ and $s_{ij} = \frac{1}{w_{ij}} \overrightarrow{\mathbf{1}_{ij}}$ into (19), we can compute $\bar{t}_n = \nabla \overline{r}_n$.[15] However, if r depends on w, we must add the following first-order term, which is due to the product rule

$$
\nabla \overline{r}_n = \bar{t}_n + \underbrace{\sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} \nabla [r_{ij}]_n}_{\text{first-order term}} \tag{21}
$$

We provide the details for computing the gradients of two first-order quantities, Shannon entropy and the KL divergence, using this recipe in §6.2 and §6.3, respectively.

5 Algorithms

Having reduced the computation of $\overline{r}$ and $\bar{t}$ to finding derivatives of Z in §4, we now describe efficient algorithms that exploit this connection. The main algorithmic ideas used in this section are based on automatic differentiation (AD) techniques (Griewank and Walther, 2008). These are general-purpose techniques for efficiently evaluating gradients given algorithms that evaluate the functions. In our setting, the algorithm in question is an efficient procedure

[15] Note that when $w_{ij} = 0$, we can set $s_{ij} = \mathbf{0}$.

1: **def** $\mathtt{T}_1\big(w\colon \mathcal{E} \mapsto \mathbb{R}, r\colon \mathcal{E} \mapsto \mathbb{R}^R\big)$:
2: ▷ *Compute first-order total; requires* $\mathcal{O}\big(N^3 + N^2 R'\big)$ *time,* $\mathcal{O}\big(N^2 + R\big)$ *space.*
3: Compute all $\widetilde{w_{ij}}$ via (14) and (22) in $\mathcal{O}\big(N^3\big)$
4: $\overline{r} \leftarrow \sum_{(i \to j) \in \mathcal{E}} \widetilde{w_{ij}} r_{ij}$ ▷ $\mathcal{O}\big(N^2 R'\big)$
5: **return** $\overline{r}$

Figure 2: Algorithm for first-order totals.

for evaluating Z, such as the procedure we described in §2.1. While we provide derivatives §5.1 in our algorithms, these can also be evaluated using any AD library, such as JAX (Bradbury et al., 2018), PyTorch (our choice) (Paszke et al., 2019), or TensorFlow (Abadi et al., 2015).

Proposition 4 is realized as $\mathtt{T}_1$ in Fig. 2 and (19) and (20) are realized as $\mathtt{T}_2^{\mathrm{v}}$ and $\mathtt{T}_2^{\mathrm{h}}$ in Fig. 3, respectively. We provide the runtime complexity of each step in the algorithms. These will be discussed in more detail in §5.2.

5.1 Derivatives of Z

All three algorithms rely on first- or second-order derivatives of Z. Since $\mathrm{Z} = |\mathbf{L}|$, we can express its gradient via Jacobi's formula and an application of the chain rule[16]

$$
\frac{\partial \mathrm{Z}}{\partial w_{ij}} = \mathrm{Z} \sum_{(i',j') \in \mathcal{L}_{ij}} \mathrm{B}_{i'j'} \mathbf{L}'_{i'j',ij} \tag{22}
$$

where

$$
\mathbf{B} = \mathbf{L}^{-\top} \tag{23}
$$

is the transpose of $\mathbf{L}^{-1}$, $\mathbf{L}'_{i'j',ij} = \frac{\partial \mathrm{L}_{i'j'}}{\partial w_{ij}}$, and $\mathcal{L}_{ij}$ is the set of pairs where $(i', j') \in \mathcal{L}_{ij}$ means that $\mathbf{L}'_{i'j',ij} \neq 0$. We define $\mathrm{B}_{\rho j'} \stackrel{\text{def}}{=} 0$ for any $j' \in \mathcal{N}$. Koo et al. (2007) show that for any i and j, $|\mathcal{L}_{ij}| \leq 2$ in the unlabeled case, indeed, $\mathbf{L}'_{i'j',ij}$ is given by

$$
\mathbf{L}'_{i'j',ij} = \begin{cases} 1 & \textbf{if } i' \in \{1, j\}, j' = j \\ -1 & \textbf{if } i' = i, j' = j \\ 0 & \textbf{otherwise} \end{cases} \tag{24}
$$

[16] The derivative of $|\mathbf{L}|$ can also be given using the matrix adjugate, $\nabla \mathrm{Z} = \mathrm{adj}(\mathbf{L})^\top$. There are benefits to using the adjugate as it is more numerically stable and equally efficient (Stewart, 1998). In fact, any algorithm that computes the determinant can be algorithmically differentiated to obtain an algorithm for the adjugate.

Their result holds for any Laplacian encoding we gave in §2.2.[17]

The second derivative of Z can be evaluated as follows[18]

$$\frac{\partial^2 \mathrm{Z}}{\partial w_{ij} \partial w_{kl}} = \sum_{\substack{(i',j') \in \mathcal{L}_{ij} \\ (k',l') \in \mathcal{L}_{kl}}} \mathrm{L}'_{i'j',ij} \frac{\partial^2 \mathrm{Z}}{\partial \mathrm{L}_{i'j'} \partial \mathrm{L}_{k'l'}} \mathrm{L}'_{k'l',kl} \tag{25}$$

where

$$\frac{\partial^2 \mathrm{Z}}{\partial \mathrm{L}_{i'j'} \partial \mathrm{L}_{k'l'}} = \mathrm{Z}\left(\mathrm{B}_{i'j'} B_{k'l'} - \mathrm{B}_{i'l'} B_{k'j'}\right) \tag{26}$$

Note that (25) also contains a term with $\nabla^2 \mathbf{L}$ as it is derived from the product rule. Because $\mathbf{L}$ is a linear construction, its second derivative is zero and so we can drop this term.

5.2 Complexity Analysis

The efficiency of our approach is rooted in the following result from automatic differentiation, which relates the cost of gradient evaluation to the cost of function evaluation. Given a function f, we denote the number of differentiable elementary operations (e.g., +, *, /, -, cos, pow) of f by $\mathrm{Cost}\{f\}$.

Theorem 2 (Cheap Jacobian–vector Products)**.** *For any function f: $\mathbb{R}^K \mapsto \mathbb{R}^M$ and any vector $v \in \mathbb{R}^M$, we can evaluate $(\nabla f(x))^\top v \in \mathbb{R}^K$ with cost satisfying the following bound via reverse-mode AD (Griewank and Walther, 2008, page 44),*

$$\mathrm{Cost}\left\{(\nabla f(x))^\top v\right\} \leq 4 \cdot \mathrm{Cost}\{f\} \tag{27}$$

Thus, $\mathcal{O}\big(\mathrm{Cost}\{(\nabla f(x))^\top v\}\big) = \mathcal{O}(\mathrm{Cost}\{f\})$.

As a special (and common) case, Theorem 2 implies a *cheap gradient principle*: The cost of evaluating the gradient of a function of one output ($M = 1$) is as fast as evaluating the function itself.

Algorithm $\mathtt{T}_1$. The cheap gradient principle tells us that $\nabla \mathrm{Z}$ can be evaluated as quickly as Z itself, and that numerically accurate procedures for Z give rise to similarly accurate procedures for $\nabla \mathrm{Z}$. Additionally, many widely used software libraries can do this work for us, such as JAX, PyTorch, and TensorFlow. The runtime of evaluating Z is dominated by evaluating the determinant of the Laplacian matrix. Therefore, we can find both Z and $\nabla \mathrm{Z}$ in the same complexity: $\mathcal{O}\big(N^3\big)$. Line 4 of Fig. 2 is a sum over N^2 scalar–vector multiplications of size R, this suggests a runtime of $\mathcal{O}\big(N^2 R\big)$. However, in many applications, R is a sparse function. Therefore, we find it useful to consider the complexities of our algorithms in terms of the size R, and the maximum density R' of each r_{ij}. We can then evaluate Line 4 in $\mathcal{O}\big(N^2 R'\big)$, leading to an overall runtime for $\mathtt{T}_1$ of $\mathcal{O}\big(N^3 + N^2 R'\big)$. The call to $\mathtt{Z}$ uses $\mathcal{O}\big(N^2\big)$ space to store the Laplacian matrix. Computing the gradient of Z similarly takes $\mathcal{O}\big(N^2\big)$ to store. Since storing $\overline{r}$ takes $\mathcal{O}(R)$ space, $\mathtt{T}_1$ has a space complexity of $\mathcal{O}\big(N^2 + R\big)$.

Algorithm $\mathtt{T}_2^{\mathrm{v}}$. Second-order quantities ($\bar{t}$), appear to require $\nabla^2 \mathrm{Z}$ and so do not directly fit the conditions of the cheap gradient principle: the Hessian ($\nabla^2 \mathrm{Z}$) is the Jacobian of the gradient. The approach of $\mathtt{T}_2^{\mathrm{v}}$ to work around this is to make several calls to Theorem 2 for each element of $\overline{r}$. In this case, the function in question is (11), which has output dimensionality R. Computing $\nabla \overline{r}$ can thus be evaluated with R calls to reverse-mode AD, requiring $\mathcal{O}\big(R(N^3 + N^2 R')\big)$ time. We can somewhat support fast accumulation of S'-sparse S in the summation of $\mathtt{T}_2^{\mathrm{v}}$ (Line 6). Unfortunately, $\frac{\partial \overline{r}}{\partial w_{ij}}$ will generally be dense, so the cost of the outer product on Line 6 is $\mathcal{O}(RS')$. Thus, $\mathtt{T}_2^{\mathrm{v}}$ has an overall runtime of $\mathcal{O}\big(R(N^3 + N^2 R') + N^2 R\, S'\big)$.[19] Additionally, $\mathtt{T}_2^{\mathrm{v}}$ requires $\mathcal{O}\big(N^2 R + R\, S\big)$ of space because $\mathcal{O}\big(N^2 R\big)$ is needed to compute and store the Jacobian of $\overline{r}$ and $\bar{t}$ has size $\mathcal{O}(R\, S)$.

Algorithm $\mathtt{T}_2^{\mathrm{h}}$. The downside of $\mathtt{T}_2^{\mathrm{v}}$ is that no work is shared between the R evaluations of the loop on Line 3. For our computation of Z, it turns out that substantial work can be shared among evaluations. Specifically, $\nabla^2 \mathrm{Z}$ only relies on the inverse of the Laplacian matrix, as seen in (26), leading to an alternative algorithm for second-order quantities, $\mathtt{T}_2^{\mathrm{h}}$. This is essentially the same observation made in Druck and Smith (2009). Exploiting this allows us to compute $\nabla^2 \mathrm{Z}$ in $\mathcal{O}\big(N^4\big)$ time. Note that this runtime is only achievable due to the sparsity of $\nabla \mathbf{L}$. The accumulation component (Line 12) of $\mathtt{T}_2^{\mathrm{h}}$ can be done

[17] We have that $|\mathcal{L}_{ij}| \leq 2|\mathcal{Y}|$ in the labeled case.

[18] We provide a derivation in Appendix A. Druck and Smith (2009) give a similar derivation for the Hessian, which we have generalized to any second-order quantity.

[19] If $S < R$, we can change the order of $\mathtt{T}_2^{\mathrm{v}}$ to compute $\bar{t}^\top$ in $\mathcal{O}\big(S(N^3 + N^2 S') + N^2 R' S\big)$.

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

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

  for $n = 1 \dots R$: ▷ $\mathcal{O}\big(R(N^3+N^2R')\big)$

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

  ▷ *Apply (19); requires* $\mathcal{O}\big(N^2RS'\big)$

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

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

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

Compute all $\widetilde{w_{ij}}$ using (14) and (22)

Compute all $\widetilde{w_{ij,kl}}$ using (17) and (25)

▷ *Apply (20); requires* $\mathcal{O}\big(N^4R'S'\big)$

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

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

▷ *Unified algorithm for computing second-order total; requires* $\mathcal{O}\big(N^3(R'+S')+RS+N^2\overline{R}\,\overline{S}\big)$ *time,* $\mathcal{O}\big(RS+N^2(\overline{R}+\overline{S})\big)$ *space*

▷ *The following quantities are computed in* $\mathcal{O}\big(N^3\big)$

Compute all $\widetilde{w_{ij}}$ using (14) and (22)

Compute $\mathbf{B}$ and $\mathbf{L}'$ using (23) and (24)

▷ $\overline{r}$, $\overline{s}$, *and* $\overline{f}$ *are first-order quantities.*

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

$\overline{s} \leftarrow \sum_{(i\to j)\in\mathcal{E}} \widetilde{w_{ij}} s_{ij}$ ▷ $\mathcal{O}\big(N^2S'\big)$

$\overline{f} \leftarrow \sum_{(i\to j)\in\mathcal{E}} \widetilde{w_{ij}} r_{ij} s_{ij}{}^\top$ ▷ $\mathcal{O}\big(N^2R'S'\big)$

$\widehat{r} \leftarrow \mathbf{0}; \widehat{s} \leftarrow \mathbf{0}$

for $i, j, k \in \mathcal{N}$: ▷ $\mathcal{O}\big(N^3\big)$

  for $(i', j') \in \mathcal{L}_{ij}$: ▷ $\mathcal{O}(1)$

    $\widehat{r_{kj'}}$ += $\mathrm{B}_{i'k}\mathrm{L}'_{i'j',ij} w_{ij} r_{ij}$ ▷ $\mathcal{O}(R')$

    $\widehat{s_{j'k}}$ += $\mathrm{B}_{i'k}\mathrm{L}'_{i'j',ij} w_{ij} s_{ij}$ ▷ $\mathcal{O}(S')$

▷ *Apply (32); requires* $\mathcal{O}\big(RS+N^2\overline{R}\,\overline{S}\big)$

return $\overline{f} + \frac{1}{\mathrm{Z}}\overline{r}\,\overline{s}^\top - \mathrm{Z}\sum_{j',l'\in\mathcal{N}} \widehat{r_{j'l'}}\,\widehat{s_{j'l'}}{}^\top$

Figure 3: Three algorithms for computing second-order totals. We recommend $\mathtt{T}_2$ as it achieves the best runtime in general. The algorithms $\mathtt{T}_2^{\mathrm{v}}$ and $\mathtt{T}_2^{\mathrm{h}}$ are presented for pedagogical purposes in §5.2.

in $\mathcal{O}\big(N^4R'S'\big)$. Considering space complexity, while not prevalent in our pseudocode, a benefit of $\mathtt{T}_2^{\mathrm{h}}$ is that we do not need to materialize the Hessian of Z as it only makes use of the inverse of the Laplacian matrix. Therefore, we only need $\mathcal{O}\big(N^2\big)$ space for the Laplacian inverse and $\mathcal{O}(RS)$ space for $\overline{t}$. Consequently, the $\mathtt{T}_2^{\mathrm{h}}$ requires $\mathcal{O}\big(N^2+RS\big)$ space.

Algorithm $\mathtt{T}_2$. So far we have seen that when R is small, that $\mathtt{T}_2^{\mathrm{v}}$ can be much faster than $\mathtt{T}_2^{\mathrm{h}}$. On the other hand, when R is large and $R' \ll R$, $\mathtt{T}_2^{\mathrm{h}}$ can be much faster than $\mathtt{T}_2^{\mathrm{v}}$. Can we get the best of $\mathtt{T}_2^{\mathrm{v}}$ and $\mathtt{T}_2^{\mathrm{h}}$? Our unified algorithm, $\mathtt{T}_2$ in Fig. 3, does just that. To derive it, we refactor the bottleneck of $\mathtt{T}_2^{\mathrm{h}}$ using (25) and the distributive property[20]

$$\sum_{\substack{(i\to j)\in\mathcal{E}\\(k\to l)\in\mathcal{E}}} \frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\partial w_{kl}} w_{ij} w_{kl} r_{ij} s_{kl}{}^\top = \frac{1}{\mathrm{Z}}\overline{r}\,\overline{s}^\top - \mathrm{Z}\sum_{j',l'\in\mathcal{N}} \widehat{r_{j'l'}}\,\widehat{s_{j'l'}}{}^\top \tag{28}$$

where

$$\widehat{r_{j'l'}} = \sum_{(k\to l)\in\mathcal{E}} \sum_{k'\in\mathcal{N}} \mathrm{B}_{k'j'}\mathrm{L}'_{k'l',kl} w_{kl} r_{kl} \tag{29}$$

$$\widehat{s_{j'l'}} = \sum_{(i\to j)\in\mathcal{E}} \sum_{i'\in\mathcal{N}} \mathrm{B}_{i'l'}\mathrm{L}'_{i'j',ij} w_{ij} s_{ij} \tag{30}$$

The remainder of $\overline{t}$ is given by

$$\overline{f} \stackrel{\text{def}}{=} \sum_{(i\to j)\in\mathcal{E}} \widetilde{w_{ij}} r_{ij} s_{ij}{}^\top \tag{31}$$

Therefore, we can find $\overline{t}$ by

$$\overline{t} = \overline{f} + \frac{1}{\mathrm{Z}}\overline{r}\,\overline{s}^\top - \mathrm{Z}\sum_{j',l'\in\mathcal{N}} \widehat{r_{j'l'}}\,\widehat{s_{j'l'}}{}^\top \tag{32}$$

We provide a proof in App. B.

Now, we can compute $\overline{r}$ and $\overline{s}$ using $\mathtt{T}_1$ in $\mathcal{O}\big(N^3+N^2(R'+S')\big)$ and their outer product in $\mathcal{O}(RS)$. Additionally, we can compute all $\widehat{r_{j'l'}}$ and $\widehat{s_{j'l'}}$ values in $\mathcal{O}\big(N^3R'\big)$ and $\mathcal{O}\big(N^3S'\big)$, respectively. If r is R' sparse, then each $\widehat{r_{j'l'}}$ is $\overline{R} \stackrel{\text{def}}{=} \min(R, N\,R')$ sparse. We can compute the sum over all $\widehat{r_{j'l'}}\widehat{s_{j'l'}}{}^\top$ in $\mathcal{O}\big(N^2\overline{R}\,\overline{S}\big)$ time. Combining these runtimes, we have that $\mathtt{T}_2$ runs in $\mathcal{O}\big(N^3(R'+S')+RS+N^2\overline{R}\,\overline{S}\big)$. $\mathtt{T}_2$ requires a total of $\mathcal{O}\big(RS+N^2(\overline{R}+\overline{S})\big)$: $\mathcal{O}(RS)$ space for $\overline{t}$, and $\mathcal{O}\big(N^2(\overline{R}+\overline{S})\big)$ space for the $\widehat{r}$ and $\widehat{s}$ values.

[20] Refactoring sum–product expressions via the distributive property is the cornerstone of dynamic programming; similar examples in natural language processing include Eisner and Blatz (2007) and Gildea (2011).

We return to our original question: Can we get the best of $\mathtt{T}_2^{\mathrm{v}}$ and $\mathtt{T}_2^{\mathrm{h}}$? In the case when R is small, $\mathtt{T}_2$ matches the runtime of $\mathtt{T}_2^{\mathrm{v}}$. Furthermore, in the case when R is large and $R' \ll R$, $\mathtt{T}_2$ matches the runtime of $\mathtt{T}_2^{\mathrm{h}}$. Therefore, $\mathtt{T}_2$ is able to achieve the best runtime regardless of the functions r and s.

6 Applications and Prior Work

In this section, we apply our framework to compute a number of important quantities that are used when working with probabilistic models. We relate our approach to existing algorithms in the literature (where applicable), and mention existing and potential applications. Many of our quantities were covered in Li and Eisner (2009) for B-hypergraphs; we extend their results to spanning trees.

In most applications that involve training a probabilistic model, the edge weights in the model will be parameterized in some fashion. Traditional approaches (Koo et al., 2007; Smith and Smith, 2007; McDonald et al., 2005a; Druck, 2011) use log-linear parameterizations, whereas more recent work (Dozat and Manning, 2017; Liu and Lapata, 2018; Ma and Xia, 2014) use neural-network parameterizations. Our algorithms are agnostic as to how edges are parameterized.

6.1 Risk

Risk minimization is a technique for training structured prediction models (Li and Eisner, 2009; Smith and Eisner, 2006; Stoyanov and Eisner, 2012). Risk is the expectation of a cost function $r\colon \mathcal{D} \mapsto \mathbb{R}$ that measures the number of mistakes in comparison to a target tree d^*. In the context of dependency parsing, $r(d)$ can be the labeled or unlabeled attachment score (LAS and UAS, respectively), both of which are additively decomposable. The unlabeled case decomposes as follows:

$$r_{ij} = \begin{cases} \frac{1}{N} & \textbf{if } (i \to j) \in d^* \\ 0 & \textbf{otherwise} \end{cases} \tag{33}$$

where d^* is the gold tree and N is the length of the sentence. Note that the use of $\frac{1}{N}$ ensures that $r(d)$ will be a score between 0 and 1. We can then obtain the expected attachment score using $\mathtt{T}_1$, and we can evaluate its gradient in the same run-time using reverse-mode AD or $\mathtt{T}_2$. In this case, $s : \mathcal{D} \mapsto \mathbb{R}^S$ is the one-hot representation of the edges; thus, we have $S = N^2$. However, because s is 1-sparse, we have $S' = 1$. Additionally, as r does not depend on w, we do not need to add a first-order term to find the gradient. Therefore, the runtime for the gradient is also $\mathcal{O}(N^3)$.

6.2 Shannon Entropy

Entropy is a useful measure of uncertainty, which has been used a number of times in dependency parsing (Smith and Eisner, 2007; Druck and Smith, 2009; Ma and Xia, 2014) for semi-supervised learning. Smith and Eisner (2007) employ entropy regularization (Grandvalet and Bengio, 2004) to bootstrap dependency parsing. However, they give an algorithm for the Shannon entropy,

$$\mathrm{H}(p) \stackrel{\text{def}}{=} \mathbb{E}_d[-\log p(d)] \tag{34}$$

that runs in $\mathcal{O}(N^4)$.[21] Recall from §3 that $-\log p(d)$ is additively decomposable; thus, running $\mathtt{T}_1$ with $r_{ij} = \frac{1}{N}\log \mathrm{Z} - \log w_{ij}$ computes $\mathrm{H}(p)$ in $\mathcal{O}(N^3)$. Martins et al.'s (2010) algorithm for computing $\mathrm{H}(p)$ is precisely the same as ours. However, they do not describe how to compute its gradient. As with risk, we can find the gradient of entropy using $\mathtt{T}_2$ or using reverse-mode AD. When using $\mathtt{T}_2$, since the gradient of r with respect to w is not 0, we add the first-order quantity $\mathtt{T}_1(w, \nabla r)$ as in (21). For entropy, we have that $\nabla r_{ij} = \frac{1}{N\mathrm{Z}}\nabla \mathrm{Z} - \frac{1}{w_{ij}}\overrightarrow{\mathbf{1}}_{ij}$.

Experiment. We briefly demonstrate the practical speed-up over Smith and Eisner's (2007) $\mathcal{O}(N^4)$ algorithm. We compare the average runtime per sentence of five different UD corpora.[22] The languages have different average sentence lengths to demonstrate the extra speed-up gained when calculating the entropy of longer sentences (that is, $\mathcal{D}$ would be a larger set). Tab. 1 shows that even for a corpus of short sentences (Finnish), we achieve a 4 times speed-up. This increases to 15 times as we move to corpora with longer sentences (Arabic).

6.3 Kullback–Leibler Divergence

To the best of our knowledge, no algorithms to compute the Kullback–Leibler (KL) divergence between two graph-based parsers (nor its gradient) have been given in the literature. We show how

[21] Their algorithm calls MTT N times, where the i^{th} call to MTT multiplies the set of incoming edges to i^{th} non-root node by their log weight.

[22] Times were measured using an Intel(R) Core(TM) i7-7500U processor with 16GB RAM.

Language	Sentence length	Entropy (nats / word)	Average Runtime (ms)		Speed-up
			$\mathtt{T}_1$ (Fig. 2)	Past Approach	
Finnish	9.23	0.6092	0.4623	1.882	4.1
English	12.45	0.8264	0.5102	2.778	5.4
German	17.56	0.8933	0.5583	4.104	7.3
French	24.65	0.8923	0.5635	5.742	10.2
Arabic	36.05	0.7163	0.6220	9.368	15.1

Table 1: Average runtime of computing entropy of dependency parser output on five languages. We use the weights of the Stanford Dependency Parser (Qi et al., 2018). The past approach is that of Smith and Eisner (2007).

this can be achieved easily within our framework. The KL divergence is defined as

$$\mathrm{KL}(p \,\|\, q) \stackrel{\text{def}}{=} \sum_{d \in \mathcal{D}} p(d) \log \frac{p(d)}{q(d)} \tag{35}$$

This takes a similar form to the Shannon entropy in (34). We can therefore choose our additively decomposable function to be $r_{ij} = \log \frac{w_{ij}}{q_{ij}} - \frac{1}{N} \log \mathrm{Z}$. Running $\mathtt{T}_1$ with these weights computes the KL divergence in $\mathcal{O}\big(N^3\big)$ time. To find the gradient of the KL divergence, we return the sum of $\mathtt{T}_2(w, r, s)$ where we chose $s_{ij} = \frac{1}{w_{ij}} \overrightarrow{\mathbf{1}}_{ij}$ and add $\mathtt{T}_1(w, \nabla r)$. For the KL divergence, we have that $\nabla r_{ij} = \frac{1}{w_{ij}} \overrightarrow{\mathbf{1}}_{ij} - \nabla \mathrm{Z} \frac{1}{N\mathrm{Z}}$.

6.4 Gradient of the GE Objective

The generalized expectation criterion (McCallum et al., 2007; Druck et al., 2009) is a method semi-supervised training using weakly labeled data. GE fits model parameters by encouraging models to match certain expectation constraints, such as marginal-label distributions, on the unlabeled data. More formally, let f be a feature function $f(d) \in \mathbb{R}^F$, and with a target value of $f^* \in \mathbb{R}^F$ that has been specified using domain knowledge. For example, given an English part-of-speech tagged sentence, we can provide the following light supervision to our model: determiners should attach to the nearest noun on their right. This is an example of a very precise heuristic for dependency parsing English that has high precision.

GE then minimizes the following objective,

$$\mathrm{GE}(p, f^*) = \frac{1}{2} \Big|\Big| \mathbb{E}_d[f(d)] - f^* \Big|\Big|^2 \tag{36}$$

which encourages the model parameters to match the target expectations. Most methods for optimizing (36) will make use of the gradient.

We note that by application of the chain rule, the gradient of the GE objective is a second-order quantity, and so we can use $\mathtt{T}_2$ to compute it. As we discussed in §1, the gradient of the GE has led to confusion in the literature (Druck et al., 2009; Druck and Smith, 2009; Druck, 2011). The best runtime bound prior to our work is Druck et al. (2009)'s $\mathcal{O}\big(N^4F'\big)$ algorithm. $\mathtt{T}_2$ is strictly better at $\mathcal{O}\big(N^3{+}N^2F'\big)$ time.[23] Alternatively, as the GE objective is a scalar, we can compute its gradient in $\mathcal{O}\big(N^3{+}N^2F'\big)$ using reverse-mode AD. Druck (2011) acknowledges that AD can be used, but questions its practicality and numerical accuracy. We hope to dispel this misconception in the following experiment.

Experiment. We compute the GE objective and its gradient for almost 1500 sentences of the English UD Treebank[24] (Nivre et al., 2018) using 20 features extracted using the methodology of Druck et al. (2009). We note that $\mathtt{T}_2$ obtains a speed-up of 9 times over Druck and Smith (2009)'s strategy of materializing the covariance matrix (i.e., $\mathtt{T}_2^{\mathrm{h}}$). Additionally, the gradients from both approaches are equivalent with an absolute tolerance of 10^{-16}.

7 Conclusion

We presented a general framework for computing first- and second-order expectations for additively decomposable functions. We did this by exploiting a key connection between gradients and

[23] We must apply a chain rule in order to use $\mathtt{T}_2$. To do this, we first run $\mathtt{T}_1$ to obtain $\bar{f}$ in $\mathcal{O}\big(N^3 + N^2F'\big)$. We then run $\mathtt{T}_2$ with the dot product of f and $\bar{f} - f^*$, which has a dimensionality of 1, and the sparse one-hot vectors as before. The execution of $\mathtt{T}_2$ then takes $\mathcal{O}\big(N^3\big)$, giving us the desired runtime. Full detail is available in our code.

[24] We used all sentences in the test set, which were between 5 and 150 words.

expectations that allows us to solve our problems using automatic differentiation. The algorithms we provide are simple, efficient, and extendable to many expectations. The automatic differentiation principle has been applied in other settings, such as weighted context-free grammars (Eisner, 2016) and chain-structured models (Vieira et al., 2016). We hope that this paper will also serve as a tutorial on how to compute expectations over trees so that the list of *cautionary tales* does not grow further. Particularly, we hope that this will allow for the KL divergence to be used in semi-supervised training of dependency parsers. Our aim is for our approach for computing expectations to be extended to other structured prediction models.

Acknowledgments

We would like to thank action editor Dan Gildea and the three anonymous reviewers for their valuable feedback and suggestions. The first author is supported by the University of Cambridge School of Technology Vice-Chancellor's Scholarship as well as by the University of Cambridge Department of Computer Science and Technology's EPSRC.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke,, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Eduard Bejček, Eva Hajičová, Jan Hajič, Pavlína Jínová, Václava Kettnerová, Veronika Kolářová, Marie Mikulová, Jiří Mírovský, Anna Nedoluzhko, Jarmila Panevová, Lucie Poláková, Magda Ševčíková, Jan Štěpánek, and Šárka Zikánová. 2013. Prague dependency treebank 3.0.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. 2018. JAX: Composable transformations of Python+ NumPy programs.

Adnan Darwiche. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3).

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the International Conference on Learning Representations*.

Gregory Druck. 2011. *Generalized Expectation Criteria for Lightly Supervised Learning*. Ph.D. thesis, University of Massachusetts Amherst.

Gregory Druck, Gideon Mann, and Andrew McCallum. 2009. Semi-supervised learning of dependency parsers using generalized expectation criteria. In *Proceedings of the International Joint Conference on Natural Language Processing*.

Gregory Druck and David Smith. 2009. Computing conditional feature covariance in non-projective tree conditional random fields. Technical Report UM-CS-2009-060, University of Massachusetts.

Jean-Guillaume Dumas and Victor Pan. 2016. Fast matrix multiplication and symbolic computation. *arXiv preprint arXiv:1612.05766*.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP@EMNLP 2016, Austin, TX, USA, November 5, 2016*.

Jason Eisner and John Blatz. 2007. Program transformations for optimization of parsing algorithms and other weighted logic programs. In *Proceedings of the Conference on Formal Grammar*, pages 45–85, CSLI Publications.

Harold N. Gabow and Robert Endre Tarjan. 1984. Efficient algorithms for a family of matroid intersection problems. *Journal of Algorithms*, 5(1).

Giorgio Gallo, Giustino Longo, and Stefano Pallottino. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2).

Daniel Gildea. 2011. Grammar factorization by tree decomposition. *Computational Linguistics*, 37(1):231–248.

Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*.

Andreas Griewank and Andrea Walther. 2008. *Evaluating Derivatives–Principles and Techniques of Algorithmic Differentiation*, second edition. SIAM.

M. Jerrum and M. Snir. 1982. Some exact complexity results for straight-line computations over semirings. *Journal of the Association for Computing Machinery*, 29(3).

Erich Kaltofen. 1992. On computing determinants of matrices without divisions. In *Papers from the International Symposium on Symbolic and Algebraic Computation*.

Gustav Kirchhoff. 1847. Über die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12).

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A. Smith. 2014. A dependency parser for tweets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Yang Liu and Mirella Lapata. 2018. Learning structured text representations. *Transactions of the Association for Computational Linguistics*, 6.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective MST parsing. In *Proceedings of the International Joint Conference on Natural Language Processing*.

Xuezhe Ma and Fei Xia. 2014. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

André Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mário Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44.

Andrew McCallum, Gideon Mann, and Gregory Druck. 2007. Generalized expectation criteria. University of Massachusetts.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the International Conference on Parsing Technologies*.

István Miklós. 2019. *Computational Complexity of Counting and Sampling*. CRC Press. https://www.taylorfrancis.com/books/9781315266954.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie,

Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebirŏglu Eryiĭgit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčcová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phuong Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Luong Nguyễn Thị, Huyền Nguyễn Thị. Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ. Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Rosça, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel

van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdenčk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2018. Universal dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794, Sydney, Australia. Association for Computational Linguistics.

David A. Smith and Jason Eisner. 2007. Bootstrapping feature-rich dependency parsers with entropic priors. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

David A. Smith and Noah A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

G. W. Stewart. 1998. On the adjugate matrix. *Linear Algebra and its Applications*, 283(1–3).

Veselin Stoyanov and Jason Eisner. 2012. Minimum-risk training of approximate CRF-based NLP systems. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Lucien Tesnière. 1959. *Eléments de syntaxe structurale*. Klincksieck.

W. T. Tutte. 1984. *Graph Theory*. Addison-Wesley Publishing Company.

Tim Vieira, Ryan Cotterell, and Jason Eisner. 2016. Speed-accuracy tradeoffs in tagging with variable-order CRFs and structured sparsity. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Tim Vieira and Jason Eisner. 2017. Learning to prune: Exploring the frontier of fast and accurate parsing. *Transactions of the Association for Computational Linguistics*, 5:263–278.

Martin J. Wainwright and Michael I. Jordan. 2008. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc.

Ran Zmigrod, Tim Vieira, and Ryan Cotterell. 2020. Please mind the root: Decoding arborescences for dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4809–4819.

A Derivation of $\nabla^2 \mathrm{Z}$

In this section, we will provide a derivation for the expression of $\nabla^2 \mathrm{Z}$ given in (25). We begin by taking the derivative of $\nabla \mathrm{Z}$ using (22)

$$\frac{\partial^2 \mathrm{Z}}{\partial w_{ij} \partial w_{kl}} = \frac{\partial}{\partial w_{ij}} \mathrm{Z} \sum_{(k',l') \in \mathcal{L}_{kl}} \mathrm{B}_{k'l'} \mathrm{L}'_{k'l',kl}$$

We solve this by applying the product rule.[25] The first term of the product rule is

$$\begin{aligned} &\frac{\partial \mathrm{Z}}{\partial w_{ij}} \sum_{(k',l') \in \mathcal{L}_{kl}} \mathrm{B}_{k'l'} \mathrm{L}'_{k'l',kl} \\ &= \mathrm{Z} \sum_{\substack{(i',j') \in \mathcal{L}_{ij} \\ (k',l') \in \mathcal{L}_{kl}}} \mathrm{B}_{i'j'} \mathrm{B}_{k'l'} \mathrm{L}'_{i'j',ij} \mathrm{L}'_{k'l',kl} \end{aligned}$$

[25] Note that we do not have to take the derivative of $\mathrm{L}'_{k'l',kl}$ as it is either 1 or -1.

The second term of the product rule is

$$\begin{aligned}&\mathrm{Z}\sum_{(k',l')\in\mathcal{L}_{kl}}\frac{\partial \mathrm{B}_{k'l'}}{\partial w_{ij}}\mathrm{L}'_{k'l',kl}\\&=-\mathrm{Z}\sum_{\substack{(i',j')\in\mathcal{L}_{ij}\\(k',l')\in\mathcal{L}_{kl}}}\mathrm{B}_{i'l'}\mathrm{B}_{k'j'}\mathrm{L}'_{i'j',ij}\mathrm{L}'_{k'l',kl}\end{aligned}$$

Summing these together yields (25).

B Proof of $\mathtt{T}_2$

In this section, we will prove the decomposition of $\bar{t}$ that allows for the efficient factoring used in $\mathtt{T}_2$. First, recall from Proposition 7 that we may find $\bar{t}$ by

$$\begin{aligned}\bar{t}=&\sum_{(i\to j)\in\mathcal{E}}\left[\frac{\partial \mathrm{Z}}{\partial w_{ij}}w_{ij}r_{ij}{s_{ij}}^{\top}\right]+\\&\sum_{(i\to j)\in\mathcal{E}}\sum_{(k\to l)\in\mathcal{E}}\left[\frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\partial w_{kl}}w_{ij}w_{kl}r_{ij}{s_{kl}}^{\top}\right]\end{aligned}$$

The first summand is the first-order total for function ${r_{ij}s_{ij}}^{\top}$ (given as $\bar{f}$ in $\mathtt{T}_2$). We can write a sum over all edges as the sum over pairs of nodes in $\mathcal{N}$. Similarly, elements in $\mathcal{L}_{ij}$ can be considered as pairs of nodes. Therefore, unless specified otherwise, we assume all variables in the base of a summation are scoped to $\mathcal{N}$. Then, the second summand can then be rewritten

$$\begin{aligned}&\sum_{i\to j\in\mathcal{E}}\sum_{(k\to l)\in\mathcal{E}}\frac{\partial^2 \mathrm{Z}}{\partial w_{ij}\partial w_{kl}}w_{ij}w_{kl}r_{ij}{s_{kl}}^{\top}\\&=\sum_{i,j,k,l,i',j',k',l'}\mathrm{L}'_{i'j',ij}\mathrm{Z}\mathrm{B}_{i'j}\mathrm{B}_{k'l'}\mathrm{L}'_{k'l',kl}w_{ij}w_{kl}r_{ij}{s_{kl}}^{\top}\\&\qquad-\mathrm{L}'_{i'j',ij}\mathrm{Z}\mathrm{B}_{i'l'}\mathrm{B}_{k'j'}\mathrm{L}'_{k'l',kl}w_{ij}w_{kl}r_{ij}{s_{kl}}^{\top}\end{aligned}$$

By distributivity, the first term equals

$$\begin{aligned}&\mathrm{Z}\left[\sum_{i,j,i',j'}\mathrm{B}_{i'j'}\mathrm{L}'_{i'j',ij}w_{ij}r_{ij}\right]\left[\sum_{k,l,k',l'}\mathrm{B}_{k'l'}\mathrm{L}'_{k'l',kl}w_{kl}s_{kl}\right]^{\top}\\&=\frac{1}{\mathrm{Z}}\overline{r}\,\overline{s}^{\top}\end{aligned}$$

By distributivity, the second term equals

$$\begin{aligned}&\mathrm{Z}\sum_{j',l'}\Bigg[\underbrace{\sum_{k',k,l}\mathrm{B}_{k'j'}\mathrm{L}'_{k'l',kl}w_{kl}r_{kl}}_{\stackrel{\text{def}}{=}\widehat{r_{j'l'}}}\Bigg]\Bigg[\underbrace{\sum_{i',i,j}\mathrm{B}_{i',l'}\mathrm{L}'_{i'j',ij}w_{ij}s_{ij}}_{\stackrel{\text{def}}{=}\widehat{s_{j'l'}}}\Bigg]^{\top}\\&=\mathrm{Z}\sum_{j',l'}\widehat{r_{j'l'}}\,\widehat{s_{j'l'}}^{\top}\end{aligned}$$

The above decomposition assumed we sum over all i', j', k', and l' and so suggests we can compute all $\widehat{r_{j'l'}}$ and $\widehat{s_{j'l'}}$ in $\mathcal{O}\big(N^5(R'+S')\big)$. However, we can exploit the sparsity of $\nabla\mathbf{L}$ to improve this. Specifically, the follow algorithm computes $\widehat{r_{j'l'}}$ for all j', $l'\in\mathcal{N}$.

$\widehat{r_{j'l'}}\leftarrow\mathbf{0}$
  for $(k\to l)\in\mathcal{E}$ ▷ $\mathcal{O}\big(N^2\big)$
    for $(k'\to l')\in\mathcal{L}_{kl}$ ▷ $\mathcal{O}(1)$
      for $j'\in\mathcal{N}$ ▷ $\mathcal{O}(N)$
        $\widehat{r_{j'l'}}$ += $\mathrm{B}_{k'l'}\mathrm{L}'_{k'l',kl}w_{kl}r_{kl}$

Therefore, we can compute all $\widehat{r_{j'l'}}$ and $\widehat{s_{j'l'}}$ in $\mathcal{O}\big(N^3(R'+S')\big)$. Each $\widehat{r_{ij}}$ is at most $\mathcal{O}(NR')$ dense, because there are at most $\mathcal{O}(N)$ R'-sparse vectors added to it (by the inner loop). Hence, $\widehat{r_{ij}}$ is $\mathcal{O}\big(\overline{R}\big)$ sparse where $\overline{R}\stackrel{\text{def}}{=}\min(R,N\,R')$. This means that computing the sum of the outer-products of all $\widehat{r_{ij}}$ and $\widehat{s_{ij}}$ can be done in $\mathcal{O}\big(N^2\overline{R}\,\overline{S}\big)$. Then, given that we have

$$\bar{t}=\overline{f}+\frac{1}{\mathrm{Z}}\overline{r}\,\overline{s}-\mathrm{Z}\sum_{j',l'}\widehat{r_{j'l'}}\,\widehat{s_{j'l'}}^{\top}$$

We can find $\bar{t}$ in

$$\mathcal{O}\big(N^3(R'+S')+R\,S+N^2\overline{R}\,\overline{S}\big)$$