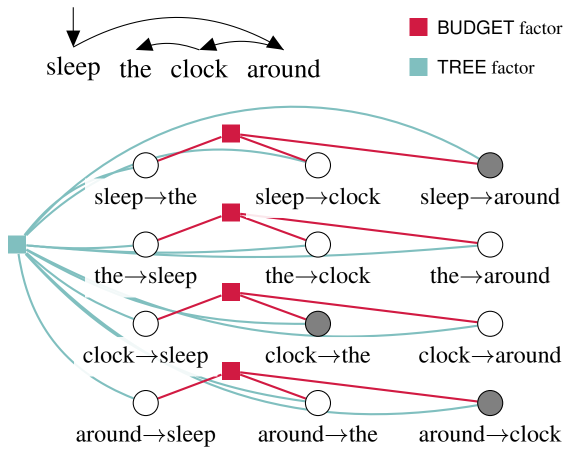
LP-SparseMAP: Differentiable Relaxed Optimization for Sparse Structured Prediction

Vlad Niculae [1] **André F. T. Martins** [1 2 3]

Abstract

Structured predictors require solving a combinatorial optimization problem over a large number of structures, such as dependency trees or alignments. When embedded as structured hidden layers in a neural net, argmin differentiation and efficient gradient computation are further required. Recently, SparseMAP has been proposed as a differentiable, sparse alternative to maximum a posteriori (MAP) and marginal inference. SparseMAP returns an interpretable combination of a small number of structures; its sparsity being the key to efficient optimization. However, SparseMAP requires access to an exact MAP oracle in the structured model, excluding, *e.g.*, loopy graphical models or logic constraints, which generally require approximate inference. In this paper, we introduce *LP-SparseMAP*, an extension of SparseMAP addressing this limitation via a local polytope relaxation. LP-SparseMAP uses the flexible and powerful language of factor graphs to define expressive hidden structures, supporting coarse decompositions, hard logic constraints, and higher-order correlations. We derive the forward and backward algorithms needed for using LP-SparseMAP as a structured hidden or output layer. Experiments in three structured tasks show benefits versus SparseMAP and Structured SVM.



```
fg = TorchFactorGraph()
u = fg.variable_from(arc_scores)
fg.add(DepTree(u))
for k in range(n):
    fg.add(Budget(u[:, k], budget=5))
fg.solve()
```

Figure 1: Parsing model with valency constraints: each "head" word is constrained to have at most k "modifiers". LP-SparseMAP is the first method for tractable, differentiable decoding in such a model. Below: abridged implementation using our library (more in App. F).

1. Introduction

The data processed by machine learning systems often has underlying structure: for instance, language data has inter-word dependency trees, or alignments, while image data can reveal object segments. As downstream models benefit from the hidden structure, practitioners typically resort to *pipelines*, training a structure predictor on labelled data, and using its output as features. This approach requires annotation, suffers from error propagation, and cannot allow the structure predictor to adapt to the downstream task.

Instead, a promising direction is to treat structure as latent, or **hidden**: learning a structure predictor without supervision, together with the downstream model in an end-to-end fashion. Several recent approaches were proposed to tackle this, based on differentiating through marginal inference (Kim et al., 2017; Liu and Lapata, 2018), noisy gradient estimates (Peng et al., 2018; Yogatama et al., 2017), or both (Corro and Titov, 2019a;b). The work in this area requires specialized, structure-specific algorithms either for comput-

[1]Instituto de Telecomunicações, Lisbon, Portugal [2]Unbabel, Lisbon, Portugal [3]Instituto Superior Técnico, University of Lisbon, Portugal. Correspondence to: Vlad Niculae <vlad@vene.ro>, André F. T. Martins <andre.t.martins@tecnico.ulisboa.pt>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

ing gradients or for sampling, limiting the choice of the practitioner to a catalogue of supported types structure. A slightly more general approach is **SparseMAP** (Niculae et al., 2018), which is differentiable and outputs combinations of a small number of structures, requiring only an algorithm for finding the highest-scoring structure (maximum a posteriori, or MAP). When increased expressivity is required, for instance through logic constraints or higher-order interactions, the search space becomes much more complicated, and MAP is typically intractable. For example, adding constraints on the depth of a parse tree makes the problems NP-hard. Our work improves the hidden structure modeling freedom available to practitioners, as follows.

- We propose a generic method for differentiable structured hidden layers, based on the flexible domain-specific language of **factor graphs**, familiar to many structured prediction practitioners.
- We derive an efficient and globally-convergent ADMM algorithm for the forward pass.
- We prove a compact, efficient form for the backward pass, reusing quantities precomputed in the forward pass, without having to unroll a computation graph.
- Our overall method is **modular**: new factor types can be added to our toolkit just by providing a MAP oracle, invoked by the generic forward and backward pass.
- The generic approach may be overridden when specialized algorithms are available. We derive efficient specialized algorithms for core building block factors such as pairwise, logical OR, negation, budget constraints, *etc.*, ensuring our toolkit is expressive *out-of-the-box*.

We show empirical improvements on inducing latent trees on arithmetic expressions, bidirectional alignments in natural language inference, and multilabel classification. Our library, with C++ and python frontends, is available at `https://github.com/deep-spin/lp-sparsemap`.

2. Background

2.1. Notation

We denote scalars, vectors and matrices as a, $\boldsymbol{a}$, and $\boldsymbol{A}$, respectively. The set of indices $\{1,\dots,d\}$ is denoted $[d]$. The Iverson bracket $[\![C]\!]$ takes the value 1 if the condition C is true, otherwise 0. The indicator vector $\boldsymbol{e}_i$ is defined as $[\boldsymbol{e}_i]_k := [\![i=k]\!]$. The i^{th} column of matrix $\boldsymbol{A}$ is $\boldsymbol{a}_i$. The canonical simplex is $\triangle := \{\boldsymbol{p}\in\mathbb{R}^d : \langle \mathbf{1},\boldsymbol{p}\rangle = 1, \boldsymbol{p}\geq \mathbf{0}\}$, and the convex hull is $\operatorname{conv}\{\boldsymbol{a}_1,\dots,\boldsymbol{a}_d\} := \{\boldsymbol{A}\boldsymbol{p} : \boldsymbol{p}\in\triangle\}$. We denote row-wise stacking of $\boldsymbol{A}_i \in \mathbb{R}^{m_i\times d}$ as $[\boldsymbol{A}_1,\dots,\boldsymbol{A}_k]\in\mathbb{R}^{(\sum_i m_i)\times d}$. Particularly, $[\boldsymbol{a},\boldsymbol{b}]$ is the concatenation of two (column) vectors. Given a vector $\boldsymbol{b}\in\mathbb{R}^d$, $\operatorname{diag}(\boldsymbol{b})\in\mathbb{R}^{d\times d}$ is the diagonal matrix with $\boldsymbol{b}$ along the diagonal. Given matrices $\boldsymbol{B}_1,\dots,\boldsymbol{B}_k$ of arbitrary dimensions $\boldsymbol{B}_i\in\mathbb{R}^{m_i\times n_i}$, define the block-diagonal matrix

$$\operatorname{bdiag}(\boldsymbol{B}_1,\dots,\boldsymbol{B}_k) = \begin{bmatrix} \boldsymbol{B}_1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \boldsymbol{B}_k \end{bmatrix} \in \mathbb{R}^{\sum_i m_i \times \sum_i n_i}.$$

2.2. Tractable structured problems

Structured prediction involves searching for valid structures over a large, combinatorial space $y\in\mathcal{Y}$. We assign a vector representation $\boldsymbol{a}_y$ to each structure. For instance, we may consider structures to be joint assignments of d binary variables (corresponding to parts of the structure) and define $(\boldsymbol{a}_y)_i = 1$ if variable i is turned on in structure y, else 0. The set of *valid structures* $\mathcal{Y}$ is typically non-trivial. For example, in *matching* problems between n workers and n tasks, we have $d=n^2$ binary variables, but the only legal assignments give exactly one task to each worker, and one worker to each task.

Maximization (MAP). Given a score vector over parts $\boldsymbol{\eta}$, we assign a score $\boldsymbol{\theta}_y = \langle \boldsymbol{a}_y, \boldsymbol{\eta}\rangle$ to each structure. Assembling all $\boldsymbol{a}_y$ as columns of a matrix $\boldsymbol{A}$, the highest-scoring structure is the one maximizing

$$\max_{y\in\mathcal{Y}} \langle \boldsymbol{\eta}, \boldsymbol{a}_y\rangle = \max_{\boldsymbol{p}\in\triangle} \langle \boldsymbol{\eta}, \boldsymbol{A}\boldsymbol{p}\rangle. \tag{1}$$

$\mathcal{M}_{\boldsymbol{A}} = \operatorname{conv}\{\boldsymbol{a}_y : y\in\mathcal{Y}\}$ is called the marginal polytope (Wainwright and Jordan, 2008), and points $\boldsymbol{\mu}\in\mathcal{M}_{\boldsymbol{A}}$ are expectations $\mathbb{E}_{y\sim\boldsymbol{p}}[\boldsymbol{a}_y]$ under some $\boldsymbol{p}\in\triangle$.

In the sequel, we split $\boldsymbol{A} = [\boldsymbol{M},\boldsymbol{N}]$ such that $\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{p}$ is the output of interest, (*e.g.*, variable assignments), while $\boldsymbol{N}\boldsymbol{p}$ captures additional structures or interactions (*e.g.*, transitions in sequence tagging). We denote the corresponding division of the score vector as $\boldsymbol{\eta} = [\boldsymbol{\eta}_M, \boldsymbol{\eta}_N]$. This distinction is not essential, as we may always take $\boldsymbol{M}=\boldsymbol{A}$ and $\boldsymbol{N}=[]$ (*i.e.*, treat additional interactions as first-class variables), but it is more consistent with pairwise Markov Random Fields (MRF).

Examples. A model with 3 variables and an XOR constraint (exactly one variable may be on) has possible configurations $\boldsymbol{m}_y = \boldsymbol{e}_y$ for $y\in\{1,2,3\}$, thus $\boldsymbol{M}=\boldsymbol{I}$, and no additionals ($\boldsymbol{N}=[]$). A model with the same dimension but without the constraint has all 2^3 possible configurations as columns of $\boldsymbol{M}$, still with no additionals. One such configuration is $y=\texttt{011}$, with $\boldsymbol{m}_y=[0,1,1]$. (Throughout this paper, y is an arbitrary index type with no mathematical properties; we may as well use an integer base-2 encoding.) A *sequence* model with no constraints will have the same valid configurations, but will include additionals for transition potentials: here it is sufficient to have an additional bit for each consecutive pair of variables, assigning 1 if both variables are simultaneously active. For $y=\texttt{011}$ this gives $\boldsymbol{n}_y=[0,1]$.

Optimization as a hidden layer. Hidden layers in a neural network are vector-to-vector mappings, and learning is typically done using stochastic gradients. We may cast structured maximization in this framework. Assuming fixed tie-breaking, we may regard the MAP computation as a function that takes the scores $\boldsymbol{\eta}$ and outputs a vector of variable assignments $\boldsymbol{\mu} \in [0,1]^d$,

$$\mathsf{MAP}_{\boldsymbol{A}}(\boldsymbol{\eta}) := \boldsymbol{\mu} \quad \text{where} \quad \boldsymbol{\mu} := \boldsymbol{m}_y, \quad y = \arg\max_{y \in \mathcal{Y}} \langle \boldsymbol{\eta}, \boldsymbol{a}_y \rangle. \tag{2}$$

The solution is always a vertex in $\{0,1\}^d$, and, for almost all $\boldsymbol{\eta}$, small changes to $\boldsymbol{\eta}$ do not change what the highest-scoring structure is. Thus, wherever $\mathsf{MAP}_{\boldsymbol{A}}$ is continuous, its gradients are null, rendering it unsuitable as a hidden layer in a neural network trained with gradient-based optimization (Peng et al., 2018).

Marginal inference. In unstructured models (*e.g.*, attention mechanisms), discrete maximization has the same null gradient issue identified in the previous paragraph, thus it is commonly replaced by its relaxation $\mathsf{softmax}(\boldsymbol{x})$. Denote the Shannon entropy of a distribution $\boldsymbol{p} \in \triangle$ by $H(\boldsymbol{p}) := -\sum_j p_j \log p_j$. The structured equivalent of softmax is the entropy-regularized problem

$$\max_{\boldsymbol{p} \in \triangle} \langle \boldsymbol{\eta}, \boldsymbol{A}\boldsymbol{p} \rangle + H(\boldsymbol{p}), \tag{3}$$

whose solution is $p^\star_y \propto \exp\langle \boldsymbol{a}_y, \boldsymbol{\eta} \rangle$. This *Gibbs distribution* is dense and induces a marginal distribution over variable assignments (Wainwright and Jordan, 2008):

$$\mathsf{Marginals}_{\boldsymbol{A}}(\boldsymbol{\eta}) := \boldsymbol{\mu} \quad \text{where} \quad \boldsymbol{\mu} := \mathbb{E}_{\boldsymbol{p}^\star}[\boldsymbol{m}_y]. \tag{4}$$

While generally intractable, for certain models, such as sequence tagging, one can efficiently compute $\mathsf{Marginals}_{\boldsymbol{A}}(\boldsymbol{\eta})$ and $\nabla \mathsf{Marginals}_{\boldsymbol{A}}(\boldsymbol{\eta})$ (often, with dynamic programming, Kim et al., 2017). In many, it is intractable, *e.g.*, matching (Valiant, 1979; Taskar, 2004, Section 3.5), dependency parsing with valency constraints (McDonald and Satta, 2007).

SparseMAP (Niculae et al., 2018) is a differentiable middle ground between maximization and expectation. It is defined via the quadratic objective

$$\max_{\boldsymbol{p} \in \triangle} \langle \boldsymbol{\eta}, \boldsymbol{A}\boldsymbol{p} \rangle - \frac{1}{2}\|\boldsymbol{M}\boldsymbol{p}\|^2. \tag{5}$$

where an optimal sparse distribution $\boldsymbol{p}$ and the unique $\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{p}$ can be efficiently computed via the *active set* method (Nocedal and Wright, 1999, Ch. 16.4 & 16.5), a generalization of Wolfe's *min-norm point method* (Wolfe, 1976) and an instance of *conditional gradient* (Frank and Wolfe, 1956). Remarkably, the active set method only requires calls to a maximization oracle (*i.e.*, finding the highest-scoring structure repeatedly, after adjustments), and has linear, finite convergence. Thus, SparseMAP can be computed efficiently even when marginal inference is not available, potentially turning any structured problem with a maximization algorithm available into a differentiable sparse structured hidden layer. The sparsity not only brings computational advantages, but also aids visualization and interpretation.

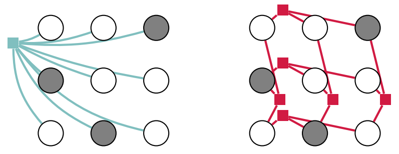


Figure 2: Matching model under two equivalent decompositions. Left: a coarse one with a single factor. Right: a fine one with multiple XOR factors.

However, the requirement of an exact maximization algorithm is still a rather stringent limitation. In the remainder of the section, we look into a flexible family of structured models where maximization is hard. Then, we extend SparseMAP to cover all such models.

2.3. Intractable structured problems and factor graph representations

We now turn to more complicated structured problems, consisting of multiple interacting subproblems. As we shall see, this covers many interesting problems.

Essentially, we represent the global structure as assignments to d variables, and posit a **decomposition** of the problem into local *factors* $f \in \mathcal{F}$, each encoding locally-tractable scoring and constraints (Kschischang et al., 2001). A factor may be seen as smaller structured subproblem. Crucially, factor must agree whenever they **overlap**, rendering the subproblems interdependent, non-separable.

Examples. Figure 1 shows a factor graph for a dependency parsing problem in which prior knowledge dictates *valency constraints*, *i.e.*, disallowing words to be assigned more than k dependent modifiers. This encourages depth, preventing trees from being too flat. For a sentence with m words, we use m^2 binary variables for every possible arc, (including the root arcs, omitted in the figure). The global tree factor disallows assignments that are not trees, and the m *budget* constraint factors, each governing $m-1$ different variables, disallow more than k dependency arcs out of each word. Factor graph representations are often **not unique**. For instance, consider a matching (linear assignment) model (Figure 2). We may employ a coarse factorization consisting of a single *matching* factor, for which maximization is tractable thanks to the Kuhn-Munkres algorithm (Kuhn, 1955). This problem can also be represented using multiple

XOR factors, constraining that each row and each column must have exactly (exclusively) one selected variable.

Denote the variable assignments as $\boldsymbol{\mu} \in [0,1]^d$. We regard each factor f as a separate structured model in its own right, encoding its permissible assignments as columns of a matrix $\boldsymbol{A}_f = [\boldsymbol{M}_f, \boldsymbol{N}_f]$, and define a *selector matrix* $\boldsymbol{C}_f$ such that $\boldsymbol{C}_f\boldsymbol{\mu}$ "selects" the variables from the global vector $\boldsymbol{\mu}$ covered by the factor f. Then, a *valid* global assignment can be represented as a tuple of local assignments y_f, provided that the agreement constraints are satisfied:

$$\mathcal{Y} = \{y = (y_f)|_{f\in\mathcal{F}} : \exists\, \boldsymbol{\mu},\, \forall f \in \mathcal{F},\, \boldsymbol{C}_f\boldsymbol{\mu} = \boldsymbol{m}_{y_f}\}. \quad (6)$$

Finding the highest scoring structure has the same form as in the tractable case, but the discrete agreement constraints in $\mathcal{Y}$ make it difficult to compute, even when each factor is computationally friendly:

$$\max_{y\in\mathcal{Y}} \sum_{f\in\mathcal{F}} \langle \boldsymbol{\eta}_f, \boldsymbol{a}_{y_f}\rangle. \quad (7)$$

In the tractable case, we were able to relax the discrete maximization into a continuous one with respect to a distribution over global configurations $\boldsymbol{p} \in \triangle$ (Eq. 1). We take the same approach, but *locally*, considering distributions over *local* configurations $\boldsymbol{p}_f \in \triangle_f$ for each factor. For compactness, we shall use the concatenations

$$\boldsymbol{p} \coloneqq [\boldsymbol{p}_{f_1}, \dots, \boldsymbol{p}_{f_n}], \quad \boldsymbol{C} \coloneqq [\boldsymbol{C}_{f_1}, \dots, \boldsymbol{C}_{f_n}]$$

and the block-diagonal matrices

$$\boldsymbol{A} \coloneqq \operatorname{bdiag}(\boldsymbol{A}_{f_1}, ..., \boldsymbol{A}_{f_n}), \boldsymbol{M} \coloneqq \operatorname{bdiag}(\boldsymbol{M}_{f_1}, ..., \boldsymbol{M}_{f_n}).$$

We may then write the optimization problem

$$\begin{aligned} \underset{\boldsymbol{\mu},\,\boldsymbol{p}}{\text{maximize}} \quad & \sum_{f\in\mathcal{F}} \langle \boldsymbol{\eta}_f, \boldsymbol{A}_f\boldsymbol{p}_f\rangle \\ \text{subject to} \quad & \boldsymbol{p} \in \triangle_{f_1} \times \triangle_{f_2} \times \dots \times \triangle_{f_n}, \\ & \boldsymbol{C}\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{p}, \end{aligned} \quad (8)$$

continuously relaxing each factor independently while enforcing agreement. The objective in Eq. 8 is separable, but the constraints are not. The feasible set,

$$\mathcal{L} = \{\boldsymbol{A}\boldsymbol{p} : \boldsymbol{p} \in \triangle_{f_1} \times \dots \times \triangle_{f_n},\, \boldsymbol{C}\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{p}\}, \quad (9)$$

is called the *local polytope* and satisfies $\mathcal{L} \supseteq \mathcal{M} = \operatorname{conv}\{\boldsymbol{a}_y : y \in \mathcal{Y}\}$. Therefore, (8) is a relaxation of (7), known as LP-MAP (Wainwright and Jordan, 2008). In general, the inclusion $\mathcal{L} \supseteq \mathcal{M}$ is strict. Many LP-MAP algorithms exploiting the graphical model structure have been proposed, from the perspective of message passing or dual decomposition (Wainwright et al., 2005; Kolmogorov, 2006; Komodakis et al., 2007; Globerson and Jaakkola, 2007; Koo et al., 2010). In particular, AD^3 (Martins et al., 2015) tackles LP-MAP by solving a SparseMAP-like quadratic subproblem for each factor.

It may be tempting to consider building a differentiable structured hidden layer by using SparseMAP with an LP-MAP approximate oracle. However, since LP-MAP is an outer relaxation, solutions are in general not feasible, leading to divergence. Instead, in the sequel, we apply the LP relaxation to a smoothed objective, resulting in a general algorithm for sparse differentiable inference.

3. LP-SparseMAP

By analogy to Eq. 5, we propose the differentiable LP-SparseMAP inference strategy:

$$\begin{aligned} \underset{\boldsymbol{\mu},\,\boldsymbol{p}}{\text{maximize}} \quad & \Big(\sum_{f\in\mathcal{F}} \langle \boldsymbol{\eta}_f, \boldsymbol{A}_f\boldsymbol{p}_f\rangle\Big) - {}^1\!/_2\, \|\boldsymbol{\mu}\|^2 \\ \text{subject to} \quad & \boldsymbol{p} \in \triangle_{f_1} \times \triangle_{f_2} \times \dots \times \triangle_{f_n}, \\ & \boldsymbol{C}\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{p}. \end{aligned} \quad (10)$$

Unlike LP-MAP (Eq. 8), LP-SparseMAP has a non-separable ℓ_2 term in the objective. The next result reformulated the problem as separable consensus optimization.

Proposition 1. *Denote by* $\deg(j) = |\{f \in \mathcal{F} : j \in f\}| > 0$, *the number of factors governing* μ_j.[1] *Define* $\boldsymbol{\delta}$ *as* $\delta_j = \sqrt{\deg(j)}$, *and* $\boldsymbol{D} = \operatorname{diag}(\boldsymbol{C}\boldsymbol{\delta})$. *Denote* $\widetilde{\boldsymbol{C}} = \boldsymbol{D}^{-1}\boldsymbol{C}, \widetilde{\boldsymbol{M}} = \boldsymbol{D}^{-1}\boldsymbol{M}$. *Then, the problem below is equivalent to* (10)*:*

$$\begin{aligned} \underset{\boldsymbol{\mu},\,\boldsymbol{p}}{\textit{maximize}} \quad & \sum_{f\in\mathcal{F}} \Big(\langle \boldsymbol{\eta}_f, \boldsymbol{A}_f\boldsymbol{p}_f\rangle - {}^1\!/_2\, \|\widetilde{\boldsymbol{M}}_f\boldsymbol{p}_f\|^2\Big) \\ \textit{subject to} \quad & \boldsymbol{p} \in \triangle_{f_1} \times \triangle_{f_2} \times \dots \times \triangle_{f_n}, \\ & \widetilde{\boldsymbol{C}}\boldsymbol{\mu} = \widetilde{\boldsymbol{M}}\boldsymbol{p}. \end{aligned} \quad (11)$$

Proof. The constraints $\boldsymbol{C}\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{p}$ and $\widetilde{\boldsymbol{C}}\boldsymbol{\mu} = \widetilde{\boldsymbol{M}}\boldsymbol{p}$ are equivalent since $\boldsymbol{\delta} > 0$ ensures $\boldsymbol{D}$ invertible. It remains to show that, at feasibility, $\|\boldsymbol{\mu}\|^2 = \|\widetilde{\boldsymbol{M}}\boldsymbol{p}\|^2$. This follows from $\|\boldsymbol{\mu}\|^2 = \|\widetilde{\boldsymbol{C}}\boldsymbol{\mu}\|^2$ (shown in App. A). □

3.1. Forward pass

Using this reformulation, we are now ready to introduce an ADMM algorithm (Glowinski and Marroco, 1975; Gabay and Mercier, 1976; Boyd et al., 2011) for maximizing Eq. 11. The algorithm is given in Algorithm 1 and derived in App. B. Like AD^3, it iterates alternating between:

1. solving a SparseMAP subproblem for each factor; (With the active set algorithm, this requires only cheap calls to a MAP oracle.)

[1] Variables not attached to any factor can be removed from the problem, so we may assume $\deg(j) > 0$.

Algorithm 1 ADMM for LP-SparseMAP

```
1: Input: $\boldsymbol{\eta}$ (scores), $T$ (max. iterations), $\gamma$ (ADMM step size), $\varepsilon_p, \varepsilon_d$ (primal and dual stopping criteria).
2: Output: $(\boldsymbol{\mu}, \boldsymbol{p})$ solving Eq. 10.
3: Initialization: $\mu_i^{(0)} = 1/\deg(i), \boldsymbol{\lambda}^{(0)} = \mathbf{0}$.
4: for $t = 1, \dots, T$
5:     for all $f \in \mathcal{F}$                    # SparseMAP subproblem
6:         $\widetilde{\boldsymbol{\eta}}_{f,M} \leftarrow \frac{1}{\gamma+1}\left(\boldsymbol{D}_f\boldsymbol{\eta}_{f,M} - \boldsymbol{\lambda}_f^{(t-1)} + \gamma\widetilde{\boldsymbol{C}}_f\boldsymbol{\mu}^{(t-1)}\right)$
7:         $\widetilde{\boldsymbol{\eta}}_{f,N} \leftarrow \frac{1}{\gamma+1}\boldsymbol{\eta}_{f,N}$
8:         $\boldsymbol{p}_f^{(t)} \leftarrow \operatorname*{arg\,min}_{\boldsymbol{p}_f\in\triangle_f} \frac{1}{2}\|\widetilde{\boldsymbol{\eta}}_{f,M} - \widetilde{\boldsymbol{M}}_f\boldsymbol{p}_f\|^2 - \langle\widetilde{\boldsymbol{\eta}_{f,N}}, \boldsymbol{N}_f\boldsymbol{p}_f\rangle$
9:     end for
10:    $\boldsymbol{\mu}^{(t)} \leftarrow \widetilde{\boldsymbol{C}}^\top\widetilde{\boldsymbol{M}}\boldsymbol{p}^{(t)}$          # agreement by local averaging
11:    $\boldsymbol{\lambda}^{(t)} \leftarrow \boldsymbol{\lambda}^{(t-1)} + \gamma\left(\widetilde{\boldsymbol{C}}\boldsymbol{\mu}^{(t)} - \widetilde{\boldsymbol{M}}\boldsymbol{p}^{(t)}\right)$          # dual update
12:    if $\|\boldsymbol{\mu}^{(t)} - \boldsymbol{\mu}^{(t-1)}\| < \varepsilon_d$ & $\|\widetilde{\boldsymbol{C}}\boldsymbol{\mu}^{(t)} - \widetilde{\boldsymbol{M}}\boldsymbol{p}^{(t)}\| < \varepsilon_p$
13:        return                                  # converged
14:    end if
15: end for
```

2. enforcing global agreement by averaging;
3. performing a gradient update on the dual variables.

Proposition 2. *Algorithm 1 converges to a solution of (10); moreover, the number of iterations needed to reach ϵ dual suboptimality is $\mathcal{O}(1/\epsilon)$.*

Proof. The algorithm is an instantiation of ADMM to Eq. 11, inheriting the proof of convergence of ADMM. (Boyd et al., 2011, Appendix A). From Proposition 1, this problem is equivalent to (10). Finally, the rate of convergence is established by Martins et al. (2015, Proposition 8), as the problems differ only through an additional regularization term in the objective. □

When there is a single factor, *i.e.*, $\mathcal{F} = \{f\}$, running for one iteration with $\gamma = 0$ recovers SparseMAP. In practice, in the inner active set solver we use warm starts and perform a small number of MAP calls. This leads to an algorithm more similar in spirit to Frank-Wolfe splitting (Gidel et al., 2018), with the key difference that by solving the nested QPs we obtain the necessary quantities to ensure a more efficient backward pass, as described in the next section.

3.2. Backward pass

Unlike marginal inference, LP-SparseMAP encourages the local distribution at each factor to become sparse, and yields a simple form for the LP-SparseMAP Jacobian, defined in terms of the local SparseMAP Jacobians of each factor (App. C.1). Denote the local solutions $\boldsymbol{\mu}_f = \widetilde{\boldsymbol{M}}\boldsymbol{p}_f$ and the Jacobians of the SparseMAP subproblem for each factor as

$$\boldsymbol{J}_{f,M} := \frac{\partial\boldsymbol{\mu}_f}{\partial\boldsymbol{\eta}_{f,M}}, \quad \boldsymbol{J}_{f,N} := \frac{\partial\boldsymbol{\mu}_f}{\partial\boldsymbol{\eta}_{f,N}}. \tag{12}$$

Algorithm 2 Backward pass for LP-SparseMAP

```
1: Input: $\boldsymbol{d}$ (the gradient of the loss w.r.t. $\boldsymbol{\mu}$), $T$ (the maximum number of iterations), $\varepsilon$ (stopping criterion).
2: Output: $\boldsymbol{d}_M, \boldsymbol{d}_{N,f}$ (loss gradient w.r.t. $\boldsymbol{\eta}_M$ and $\boldsymbol{\eta}_{N,f}$).
3: for $t = 1, \dots, T$
4:     for all $f \in \mathcal{F}$
5:         $\boldsymbol{d}_f \leftarrow \widetilde{\boldsymbol{C}}_f\boldsymbol{d}$;          # split $\boldsymbol{d}$ into copies for each factor
6:         $\boldsymbol{d}_{M,f} \leftarrow \boldsymbol{J}_{M,f}^\top\boldsymbol{d}_f$, $\boldsymbol{d}_{N,f} \leftarrow \boldsymbol{J}_{N,f}^\top\boldsymbol{d}_f$;          # local $\nabla$
7:     end for
8:     $\boldsymbol{d}_M \leftarrow \sum_f \widetilde{\boldsymbol{C}}_f^\top\boldsymbol{d}_f$.          # local averaging
9:     if $\|\boldsymbol{d}_M - \boldsymbol{d}\| \leq \varepsilon$
10:        return $(\boldsymbol{d}_M, \boldsymbol{d}_{N,f})$.          # converged
11:    else
12:        $\boldsymbol{d} \leftarrow \boldsymbol{d}_M$
13:    end if
14: end for
```

When using the active set algorithm for SparseMAP, $\boldsymbol{J}_{f,\{M,N\}}$ are precomputed in the forward pass (Niculae et al., 2018). The LP-SparseMAP backward pass combines the local Jacobians while taking into account the agreement constraints, as shown next.

Proposition 3. *Let* $\boldsymbol{J}_M = \operatorname{bdiag}(\boldsymbol{J}_{f,M})$ *and* $\boldsymbol{J}_N = \operatorname{bdiag}(\boldsymbol{J}_{f,N})$ *denote the block-diagonal matrices of local SparseMAP Jacobians. Let* $\boldsymbol{J} = \boldsymbol{J}^\top \in \mathbb{R}^{d\times d}$ *satisfying*

$$\boldsymbol{J} := \widetilde{\boldsymbol{C}}^\top\boldsymbol{J}_M\widetilde{\boldsymbol{C}}\,\boldsymbol{J}. \tag{13}$$

$$\textit{Then,}\quad \frac{\partial\boldsymbol{\mu}}{\partial\boldsymbol{\eta}_M} = \boldsymbol{J} \quad\textit{and}\quad \frac{\partial\boldsymbol{\mu}}{\partial\boldsymbol{\eta}_N} = \boldsymbol{J}\widetilde{\boldsymbol{C}}^\top\boldsymbol{J}_N. \tag{14}$$

The proof is given in App. C.2, and $\boldsymbol{J}$ may be computed using an eigensolver. However, to use LP-SparseMAP as a hidden layer, we don't need a materialized Jacobian, just its multiplication by an arbitary vector $\boldsymbol{d} \in \mathbb{R}^d$, *i.e.*,

$$\left(\frac{\partial\boldsymbol{\mu}}{\partial\boldsymbol{\eta}_M}\right)^\top\boldsymbol{d}, \qquad\text{and}\qquad \left(\frac{\partial\boldsymbol{\mu}}{\partial\boldsymbol{\eta}_N}\right)^\top\boldsymbol{d}.$$

These can be computed iteratively by Algorithm 2. Since $\boldsymbol{C}_f$ are highly sparse and structured selector matrices, lines 5 and 8 are fast indexing operations followed by scaling; the bulk of the computation is line 6, which can be seen as **invoking the backward pass of each factor**, as if that factor were alone in the graph. The structure of Algorithm 2 is similar to Algorithm 1, however, our backward is much more efficient than "unrolling" Algorithm 1 within a computation graph: Our algorithm only requires access to the final state of the ADMM solver (Algorithm 1), rather than all intermediate states, as would be required for unrolling.

3.3. Implementation and specializations

The forward and backward passes of LP-SparseMAP, described above, are appealing from the perspective of modu-

Table 1: Examples of logic constraint factors.

name	constraints
XOR (exactly one)	$\sum_{i=1}^{d} \mu_i = 1$
AtMostOne	$\sum_{i=1}^{d} \mu_i \leq 1$
OR	$\sum_{i=1}^{d} \mu_i \geq 1$
BUDGET	$\sum_{i=1}^{d} \mu_i \leq B$
Knapsack	$\sum_{i=1}^{d} c_i \mu_i \leq B$
OROut	$\sum_{i=1}^{d-1} \mu_i \geq \mu_d; \mu_i \leq \mu_d$ for all i

lar implementation. The outer loop interacts with a factor with only two interfaces: a `SolveSparseMAP` function and a `JacobianTimesVector` function. In turn, both methods can be implemented in terms of a `SolveMAP` maximization oracle (Niculae et al., 2018).

For certain factors, such as the logic constraints in Table 1, faster direct implementations of `SolveSparseMAP` and `JacobianTimesVector` are available, and our algorithm easily allows specialization. This is appealing from a testing perspective, as the specializations must agree with the generic implementation. For example, the exclusive-or XOR factor requires that exactly one out of d variables can be on. Its marginal polytope is the convex hull of allowed assignments, $\mathcal{M}_{\text{XOR}} = \operatorname{conv}\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\} = \triangle^d$. The required SparseMAP subproblem with degree corrections is

$$\begin{aligned} &\text{minimize } {}^1\!/_2 \, \|\boldsymbol{\mu} - \boldsymbol{\eta}\|_2^2 \\ &\text{subject to } \sum_{j=1}^{d} \delta_j \mu_j = 1, \text{ and } 0 \leq \mu_i \leq {}^1\!/_{\delta_i}. \end{aligned} \tag{15}$$

When $\boldsymbol{\delta} = \mathbf{1}$ this is a projection onto the simplex (sparsemax), for which efficient algorithms are well-studied (Martins and Astudillo, 2016). For general $\boldsymbol{\delta}$, the algorithm of Pardalos and Kovoor (1990) applies, and the backward pass involves a generalization of the sparsemax Jacobian.

In App. D, we derive specialized forward and backward passes for XOR, and the constraint factors in Table 1, as well as for negated variables, OR, OR-Output, Knapsack and pairwise (Ising) factors.

4. LP-SparseMAP loss for structured outputs

So far, we described LP-SparseMAP for structured hidden layers. When supervision is available, either as a downstream objective or as partial supervision, a natural convex loss relaxes the SparseMAP loss (Niculae et al., 2018):

$$\ell(\boldsymbol{\eta}, y) := \max_{\boldsymbol{p},\boldsymbol{\mu}} \sum_f \langle \boldsymbol{A}_f^\top \boldsymbol{\eta}_f, \boldsymbol{p}_f - \boldsymbol{e}_{y_f} \rangle + \frac{1}{2}(\|\boldsymbol{m}_y\|^2 - \|\boldsymbol{\mu}\|^2), \tag{16}$$

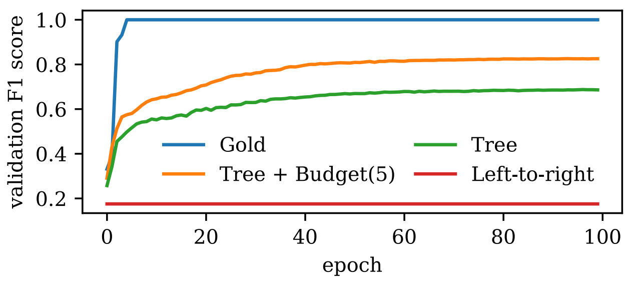


Figure 3: F_1 score for tagging ListOps nodes with their valency, using a latent tree. Incorporating inductive bias via budget constraints improves performance.

under the constraints of Eq. 10. Like the SparseMAP loss, this LP-SparseMAP loss falls into the recently-proposed class of Fenchel-Young losses (Blondel et al., 2019), which confirms its convenient properties, notably the *margin* property (Blondel et al., 2020, Proposition 8). Its gradients are obtained from the LP-SparseMAP solution $(\boldsymbol{\mu}, \boldsymbol{p})$ as

$$\nabla_{\boldsymbol{\eta}_M} \ell(\boldsymbol{\eta}, y) = \boldsymbol{\mu} - \boldsymbol{m}_y, \tag{17}$$

$$\nabla_{\boldsymbol{\eta}_f, N} \ell(\boldsymbol{\eta}, y) = \boldsymbol{N}_f \boldsymbol{p}_f - \boldsymbol{n}_{y_f}. \tag{18}$$

When already using LP-SparseMAP as a hidden layer, this loss provides a natural way to incorporate supervision on the latent structure at no additional cost.

5. Experiments

In this section, we demonstrate LP-SparseMAP for learning complex latent structures on both toy and real-world datasets, as well as on a structured output task. Learning hidden structures solely from a downstream objective is challenging for powerful models that can bypass the latent component entirely. For this reason, we design our experiments using simpler, smaller networks where the inferred structure is an un-bypassable *bottleneck*, ensuring the predictions depend on it. We use Dynet (Neubig et al., 2017) and list hyperparameter configurations and ranges in App. E.

5.1. ListOps valency tagging

The ListOps dataset (Nangia and Bowman, 2018) is a synthetic collection of bracketed expressions, such as `[max 2 9 [min 4 7 ] 0 ]`. The arguments are lists of integers, and the operators are set summarizers such as `median`, `max`, `sum`, etc. It was proposed as a litmus test for studying latent tree learning models, since the syntax is essential to the semantics. Instead of tackling the challenging task of learning to *evaluate* the expressions, we follow Corro and Titov (2019b) and study a *tagging* task: labeling each operator with the number of arguments it governs.

Table 2: ListOps tagging results with non-projective latent trees. The budget constraints bring improvement.

	validation		test	
	Acc.	F_1	Acc.	F_1
left-to-right	28.14	17.54	28.07	17.43
tree	68.23	68.74	68.74	69.12
tree+budget	**82.35**	**82.59**	**82.75**	**82.95**

Model architecture. We encode the sequence with a BiLSTM, yielding vectors $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L$. We compute the score of dependency arc $i \to j$ as the dot product between the outputs of two mappings, one for encoding the head and one for the modifier (target word):

$$\boldsymbol{f}_{\text{hd}}(\boldsymbol{h}) = \boldsymbol{W}_{\text{hd}}\boldsymbol{h} + \boldsymbol{b}_{\text{hd}}; \quad \boldsymbol{f}_{\text{mo}}(\boldsymbol{h}) = \boldsymbol{W}_{\text{mo}}\boldsymbol{h} + \boldsymbol{b}_{\text{mo}};$$

$$\eta_{i\to j} = \langle \boldsymbol{f}_{\text{hd}}(\boldsymbol{h}_i), \text{ReLU}(\boldsymbol{f}_{\text{mo}}(\boldsymbol{h}_j))\rangle.$$

We perform LP-SparseMAP optimization to get the sparse arc posterior probabilities, using different factor graph structures $\mathcal{F}$, described in the next paragraph.

$$\boldsymbol{\mu} = \text{LP-SparseMAP}_{\mathcal{F}}(\boldsymbol{\eta}) \tag{19}$$

The arc posteriors $\boldsymbol{\mu}$ correspond to a sparse combination of dependency trees. We perform one iteration of a Graph Convolutional Network (GCN) along the edges in $\boldsymbol{\mu}$. Crucially, the input to the GCN is not the BiLSTM output $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L)$ but a "de-lexicalized" sequence $(\boldsymbol{v}, \ldots, \boldsymbol{v})$ where $\boldsymbol{v}$ is a learned parameter vector, repeated L times regardless of the tokens. This forces the predictions to rely on the GCN and thus on the latent trees, preventing the model from using the global BiLSTM to "cheat". The GCN produces contextualized representations $(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_L)$ which we then pass through an output layer to predict the valency label for each operator node.

Factor graphs. Unlike Corro and Titov (2019b), who use projective dependency parsing, we consider the general non-projective case, making the problem more challenging. The MAP oracle is the maximum arborescence algorithm (Chu and Liu, 1965; Edmonds, 1967).

First, we consider a factor graph with a single non-projective TREE factor: in this case, LP-SparseMAP reduces to a SparseMAP baseline. Motivated by multiple observations that SparseMAP and similar latent structure learning methods tend to learn trivial trees (Williams et al., 2018) we next consider overlaying **constraints** in the form of BUDGET factors on top of the TREE factor. For every possible head i, we include a BUDGET factor allowing at most five of the possible outgoing arcs $(\mu_{i\to 1}, \ldots, \mu_{i\to L})$ to be selected.

Results. Figure 3 confirms that, unsurprisingly, the baseline with access to gold dependency structure quickly learns to predict perfectly, while the simple left-to-right baseline cannot progress. LP-SparseMAP with BUDGET constraints on the modifiers outperforms SparseMAP by over 10 percentage points (Table 2).

Table 3: NLI accuracy scores with structured attention. The LP-SparseMAP models perform competitively.

	SNLI		MultiNLI	
	valid	test	valid	test
softmax	84.44	84.62	70.06	69.42
matching	84.57	84.16	70.84	70.36
LP-matching	**84.70**	**85.04**	70.57	70.64
LP-sequential	83.96	83.67	**71.10**	**71.17**

5.2. Natural language inference with decomposable structured attention

We now turn to the task of natural language inference, using LP-SparseMAP to uncover hidden alignments for structured attention networks. Natural language inference is a pairwise classification task. Given a *premise* of length m, and a *hypothesis* of length n, the pair must be classified into one of three possible relationships: entailment, contradiction, or neutrality. We use the English language SNLI and MultiNLI datasets (Bowman et al., 2015; Williams et al., 2017), with the same preprocessing and splits as Niculae et al. (2018).

Model architecture. We use the model of Parikh et al. (2016) with no intra-attention. The model computes a joint attention score matrix $\boldsymbol{S}$ of size $m \times n$, where s_{ij} depends only on ith word in the premise and the jth word in the hypothesis (hence *decomposable*). For each premise word i, we apply **softmax** over the i^{th} row of $\boldsymbol{S}$ to get a weighted average of the hypothesis. Then, similarly, for each hypothesis word j, we apply softmax over the j^{th} row of $\boldsymbol{S}$ yielding a representation of the premise. From then on, each word embedding is combined with its corresponding weighted context using an affine function, the results are sum-pooled and passed through an output multi-layer perceptron to make a classification. We propose replacing the independent softmax attention with structured, joint attention, normalizing over both rows and columns *simultaneously* in several different ways, using LP-SparseMAP with scores $\eta_{ij} = s_{ij}$. We use frozen GloVe embeddings (Pennington et al., 2014), and all our models have 130k parameters (*cf.* App. E).

Factor graphs. Assume $m \leq n$. First, like Niculae et al. (2018), we consider a **matching** factor f:

$$\mathcal{M}_f = \Big\{\boldsymbol{\mu} \in [0,1]^{mn}; \sum_{j\in[n]} \mu_{ij} = 1, \sum_{i\in[m]} \mu_{ij} \leq 1\Big\}. \tag{20}$$

When $m = n$, linear maximization on this constraint set

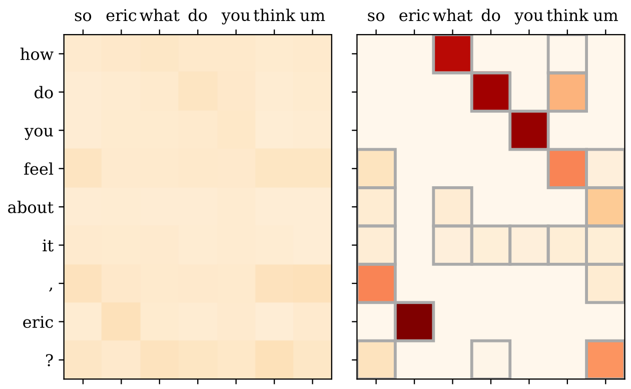


Figure 4: Attention induced using softmax (left) and LP-SparseMAP sequential (right) on a MultiNLI example. With this inductive bias, LP-SparseMAP learns a bi-directional alignment anchoring longer phrases.

corresponds to the linear assignment problem, solved by the Kuhn-Munkres (Kuhn, 1955) or Jonker-Volgenant (Jonker and Volgenant, 1987) algorithms, and the solution is a doubly stochastic matrix. When $m < n$, the scores can be padded with $-\infty$ to a square matrix prior to invoking the algorithm. A linear maximization thus takes $\mathcal{O}(n^3)$, and this instantiation of structured matching attention can be tackled by SparseMAP. Next we consider a relaxed equivalent formulation which we call **LP-matching**, as shown in Figure 2, with one XOR factor per row and one AtMostOne factor per column:

$$\begin{aligned}\mathcal{F} = \ &\{\mathsf{XOR}(\mu_{i1},\ldots,\mu_{in}) : i \in [m]\} \\ &\cup \{\mathsf{AtMostOne}(\mu_{1j},\ldots,\mu_{mj}) : j \in [n]\}\end{aligned} \tag{21}$$

Each subproblem can be solved in $\mathcal{O}(n)$ for a total complexity of $\mathcal{O}(n^2)$ per iteration (*cf.* Appendix D). While more iterations may be necessary to converge, the finer-grained approach might make faster progress, yielding more useful latent alignments. Finally, we consider a more expressive joint alignment that encourages continuity. Inspired by the sequential alignment of Niculae et al. (2018), we propose a bi-directional model called **LP-sequence**, consisting of a coarse, linear-chain Markov factor (with MAP provided by the Viterbi algorithm; Rabiner, 1989) parametrized by a single transition score η_N for every pair of alignments $(i, j)-(i+1, j\pm1)$. By itself, this factor may align multiple premise words to the same hypothesis word. We symmetrize it by overlaying m AtMostOne factors, like in Eq. 21, ensuring each hypothesis word is aligned on average to at most one premise word. Effectively, this results in a sequence tagger constrained to use each of the m states at most once. For both LP-SparseMAP approaches, we rescale the result by row sums to ensure feasibility.

Results. Table 3 reveals that LP-matching is the best performing mechanism on SNLI, and LP-sequential on MultiNLI. The η_N transition score learned by LP-sequential is 1.6 on SNLI and 2.5 on MultiNLI, and Figure 4 shows an example of the useful inductive bias it learns. On both datasets, the relaxed LP-matching outperforms the coarse matching factor, suggesting that, indeed, equivalent parametrizations of a model may perform differently when not run until convergence.

Table 4: Multilabel classification test F_1 scores.

	bibtex	bookmarks
Unstructured	42.28	35.76
Structured hinge loss	37.70	33.26
LP-SparseMAP loss	**43.43**	**36.07**

5.3. Multilabel classification

Finally, to confirm that LP-SparseMAP is also suitable as in the supervised setting, we evaluate on the task of multilabel classification. Our factor graph has k binary variables (one for each label), and a pairwise factor for every label pair:

$$\mathcal{F} = \{\mathsf{PAIR}(\mu_i, \mu_j; \eta_{ij}) : 1 \le i < j \le k\}. \tag{22}$$

This yields the standard fully-connected pairwise MRF:

$$\langle \boldsymbol{\eta}, \boldsymbol{\mu} \rangle = \sum_i \mu_i \eta_i + \sum_{i<j} \mu_i \mu_j \eta_{ij}. \tag{23}$$

Neural network parametrization. We use a 2-layer multi-layer perceptron to compute the score for each variable. In the structured models, we have an additional $1/2\, k(k-1)$ parameters for the co-occurrence score of every pair of classes. We compare an unstructured baseline (using the binary logistic loss for each label), a structured hinge loss (with LP-MAP inference) and a LP-SparseMAP loss model. We solve LP-MAP using AD^3 and LP-SparseMAP with our proposed algorithm (*cf.* Appendix E).

Results. Table 4 shows the example F_1 score on the test set for the *bibtex* and *bookmarks* benchmark datasets (Katakis et al., 2008). The structured hinge loss model is worse than the unstructured (binary logistic loss) baseline; the LP-SparseMAP loss model outperforms both. This suggests that the LP-SparseMAP loss is promising for structured output learning. We note that, in strictly-supervised setting, approaches that blend inference with learning (*e.g.*, Chen et al., 2015; Tang et al., 2016) may be more efficient; however, LP-SparseMAP can work both as a hidden layer and a loss, with no redundant computation.

6. Related work

Differentiable optimization. The most related research direction involves bi-level optimization, or *argmin differ-*

entiation (Gould et al., 2016; Djolonga and Krause, 2017); Typically, such research assumes problems are expressible in a standard form, for instance using quadratic programs (Amos and Kolter, 2017) or generic disciplined convex programs (Section 7, Amos, 2019; Agrawal et al., 2019a;b). We take inspiration from this line of work by developping LP-SparseMAP as a flexible domain-specific language for defining latent structure. The generic approaches are not applicable for the typical optimization problems arising in structured prediction, because of the intractably large number of constraints typically necessary, and the difficulty of formulating many problems in standard forms. Our method instead assumes interacting through the problem through local oracle algorithms, exploiting the structure of the factor graph and allowing for more efficient handling of coarse factors and logic constraints via *nested* subproblems.

Latent structure models. Our motivation and applications are mostly focused on learning with latent structure. Specifically, we are interested in global optimization methods, which require marginal inference or similar relaxations (Kim et al., 2017; Liu and Lapata, 2018; Corro and Titov, 2019a;b; Niculae et al., 2018), rather than incremental methods based on policy gradients (Yogatama et al., 2017). Promising methods exist for approximate marginal inference in factor graphs with MAP calls (Belanger et al., 2013; Krishnan et al., 2015; Tang et al., 2016), relying on entropy approximation penalties. Such approaches focus on supervised structure prediction, which is not our main goal; and their backward passes has not been studied to our knowledge. Importantly, as these penalties are non-quadratic, the active set algorithm does not apply, falling back to the more general variants of Frank-Wolfe. The active set algorithm is a key ingredient of our work, as it exhibits fast finite convergence, finds sparse solutions and – crucially – provides precomputation of the matrix inverse required in the backward pass (Niculae et al., 2018). In contrast, the quadratic penalty (Meshi et al., 2015; Niculae et al., 2018) is more amenable to optimization, as well as bringing other sparsity benefits. The projection step of Peng et al. (2018) can be cast as a SparseMAP problem, thus our algorithm can be used to also extend their method to arbitrary factor graphs. For pairwise MRFs (a class of factor graphs), differentiating belief propagation, either through unrolling or perturbation-based approximation, has been studied (Stoyanov et al., 2011; Domke, 2013). Our approach instead computes *implicit* gradients, which is more efficient, thanks to quantities precomputed in the forward pass, and in some circumstances has been shown to work better (Rajeswaran et al., 2019). Finally, MRF-based approaches have not been explored in the presence of logic constraints or coarse factors, while our formulation is built from the beginning with such use cases in mind.

7. Conclusions

We introduced LP-SparseMAP, an extension of SparseMAP to sparse differentiable optimization in any factor graph, enabling neural hidden layers with arbitrarily complex structure, specified using a familiar domain-specific language. We have shown LP-SparseMAP to outperform SparseMAP for latent structure learning, and outperform the structured hinge for structured output learning. We hope that our toolkit empowers future research on latent structure, leading to powerful models based on domain knowledge. In future work, we shall investigate further applications where expertise about the domain structure, together with minimal self-supervision deployed via the LP-SparseMAP loss, may lead to data-efficient learning, even for more expressive models without artificial bottlenecks.

Acknowledgements

We are grateful to Brandon Amos, Mathieu Blondel, Gonçalo Correia, Caio Corro, Erick Fonseca, Pedro Martins, Tsvetomila Mihaylova, Nikita Nangia, Fabian Pedregosa, Marcos Treviso, and the reviewers, for their valuable feedback and discussions. This work is built on open-source software; we acknowledge the scientific Python stack (Van Rossum and Drake, 2009; Oliphant, 2006; Walt et al., 2011; Virtanen et al., 2020; Behnel et al., 2011) and the developers of Eigen (Guennebaud et al., 2010). This work was supported by the European Research Council (ERC StG DeepSPIN 758969), by the Fundação para a Ciência e Tecnologia through contracts UID/EEA/50008/2019 and CMUPERI/TIC/0046/2014 (GoLocal), and by the MAIA project, funded by the P2020 program under contract number 045909.

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019a). Differentiable convex optimization layers. In *Proc. of NeurIPS*.

Agrawal, A., Barratt, S., Boyd, S., Busseti, E., and Moursi, W. M. (2019b). Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 2019(2).

Amos, B. (2019). *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University.

Amos, B. and Kolter, J. Z. (2017). OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*.

Anderson Jr, W. N., Harner, E. J., and Trapp, G. E. (1985). Eigenvalues of the difference and product of projections. *Linear and Multilinear Algebra*, 17(3-4):295–299.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: the best of both worlds. *Computing in Science & Engineering*, 13(2):31–39.

Belanger, D., Sheldon, D., and McCallum, A. (2013). Marginal inference in MRFs using Frank-Wolfe. In *Proc. of the NeurIPS Workshop on Greedy Optimization, Frank-Wolfe and Friends*.

Blondel, M., Martins, A. F., and Niculae, V. (2019). Learning classifiers with Fenchel-Young losses: Generalized entropies, margins, and algorithms. In *Proc. of AISTATS*.

Blondel, M., Martins, A. F., and Niculae, V. (2020). Learning with Fenchel-Young losses. *Journal of Machine Learning Research*, 21(35):1–69.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proc. of EMNLP*.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

Chen, L.-C., Schwing, A., Yuille, A., and Urtasun, R. (2015). Learning deep structured models. In *Proc. of ICML*.

Chu, Y.-J. and Liu, T.-H. (1965). On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Clarke, F. H. (1990). *Optimization and Nonsmooth Analysis*. SIAM.

Corro, C. and Titov, I. (2019a). Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. In *Proc. of ICLR*.

Corro, C. and Titov, I. (2019b). Learning latent trees with stochastic perturbations and differentiable dynamic programming. In *Proc. of ACL*.

Djolonga, J. and Krause, A. (2017). Differentiable learning of submodular models. In *Proc. of NeurIPS*.

Domke, J. (2013). Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467.

Edmonds, J. (1967). Optimum branchings. *J. Res. Nat. Bur. Stand.*, 71B:233–240.

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Nav. Res. Log.*, 3(1-2):95–110.

Gabay, D. and Mercier, B. (1976). A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40.

Gidel, G., Pedregosa, F., and Lacoste-Julien, S. (2018). Frank-Wolfe splitting via augmented Lagrangian method. In *Proc. of AISTATS*.

Globerson, A. and Jaakkola, T. (2007). Fixing Max-Product: Convergent message passing algorithms for MAP LP-relaxations. In *Proc. of NeurIPS*.

Glowinski, R. and Marroco, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, 9(R2):41–76.

Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., and Guo, E. (2016). On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *preprint arXiv:1607.05447*.

Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

Jonker, R. and Volgenant, A. (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340.

Katakis, I., Tsoumakas, G., and Vlahavas, I. (2008). Multilabel text classification for automated tag suggestion. In *Proc. of ECML/PKDD*.

Kim, Y., Denton, C., Hoang, L., and Rush, A. M. (2017). Structured attention networks. In *Proc. ICLR*.

Kolmogorov, V. (2006). Convergent Tree-Reweighted Message Passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583.

Komodakis, N., Paragios, N., and Tziritas, G. (2007). MRF optimization via dual decomposition: Message-Passing revisited. In *Proc. of ICCV*.

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual decomposition for parsing with non-projective head automata. In *Proc. of EMNLP*.

Krishnan, R. G., Lacoste-Julien, S., and Sontag, D. (2015). Barrier Frank-Wolfe for marginal inference. In *Proc. of NeurIPS*.

Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE T. Inform. Theory*, 47(2):498–519.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Nav. Res. Log.*, 2(1-2):83–97.

Liu, Y. and Lapata, M. (2018). Learning structured text representations. *TACL*, 6:63–75.

Martins, A. F. and Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*.

Martins, A. F., Figueiredo, M. A., Aguiar, P. M., Smith, N. A., and Xing, E. P. (2015). AD3: Alternating directions dual decomposition for MAP inference in graphical models. *JMLR*, 16(1):495–545.

McDonald, R. T. and Satta, G. (2007). On the complexity of non-projective data-driven dependency parsing. In *Proc. of ICPT*.

Meshi, O., Mahdavi, M., and Schwing, A. G. (2015). Smooth and strong: MAP inference with linear convergence. In *Proc. of NeurIPS*.

Nangia, N. and Bowman, S. (2018). ListOps: A diagnostic dataset for latent tree learning. In *Proc. of NAACL SRW*.

Neubig, G., Dyer, C., Goldberg, Y., Matthews, A., Ammar, W., Anastasopoulos, A., Ballesteros, M., Chiang, D., Clothiaux, D., Cohn, T., Duh, K., Faruqui, M., Gan, C., Garrette, D., Ji, Y., Kong, L., Kuncoro, A., Kumar, G., Malaviya, C., Michel, P., Oda, Y., Richardson, M., Saphra, N., Swayamdipta, S., and Yin, P. (2017). DyNet: The dynamic neural network toolkit. *arXiv e-prints*.

Niculae, V., Martins, A. F., Blondel, M., and Cardie, C. (2018). SparseMAP: Differentiable sparse structured inference. In *Proc. of ICML*.

Nocedal, J. and Wright, S. (1999). *Numerical Optimization*. Springer New York.

Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Omladic, M. (1987). Spectra of the difference and product of projections. *Proceedings of the American Mathematical Society*, 99:317–317.

Pardalos, P. M. and Kovoor, N. (1990). An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Mathematical Programming*, 46(1-3):321–328.

Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proc. of EMNLP*.

Parikh, N. and Boyd, S. (2014). Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239.

Peng, H., Thomson, S., and Smith, N. A. (2018). Backpropagating through structured argmax using a SPIGOT. In *Proc. of ACL*.

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proc. of EMNLP*.

Peters, B., Niculae, V., and Martins, A. F. (2019). Sparse sequence-to-sequence models. In *Proc. ACL*.

Piziak, R., Odell, P., and Hahn, R. (1999). Constructing projections on sums and intersections. *Computers & Mathematics with Applications*, 37(1):67–74.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *P. IEEE*, 77(2):257–286.

Rajeswaran, A., Finn, C., Kakade, S., and Levine, S. (2019). Meta-learning with implicit gradients. In *Proc. of NeurIPS*.

Stoyanov, V., Ropson, A., and Eisner, J. (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. of AISTATS*.

Tang, K., Ruozzi, N., Belanger, D., and Jebara, T. (2016). Bethe learning of graphical models via MAP decoding. In *Proc. of AISTATS*.

Taskar, B. (2004). *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University.

Valiant, L. G. (1979). The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189–201.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S. . . (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*.

Wainwright, M., Jaakkola, T., and Willsky, A. (2005). MAP estimation via agreement on trees: Message-Passing and Linear Programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1–2):1–305.

Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Williams, A., Drozdov, A., and Bowman, S. R. (2018). Do latent tree learning models identify meaningful structure in sentences? *TACL*.

Williams, A., Nangia, N., and Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. *preprint arXiv:1704.05426*.

Wolfe, P. (1976). Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–149.

Yogatama, D., Blunsom, P., Dyer, C., Grefenstette, E., and Ling, W. (2017). Learning to compose words into sentences with reinforcement learning. In *Proc. of ICLR*.