LEARNING DEEP MODELS WITH LINGUISTICALLY-INSPIRED STRUCTURE

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Vlad Niculae

August 2018

LEARNING DEEP MODELS WITH LINGUISTICALLY-INSPIRED STRUCTURE

Vlad Niculae, Ph.D.

Cornell University 2018

Many applied machine learning tasks involve structured representations. This is particularly the case in natural language processing (NLP), where the discrete, compositional nature of words and sentences leads to natural combinatorial representations such as trees, sequences, segments, or alignments, among others. It is no surprise that structured output models have been successful and popular in NLP applications since their inception. At the same time, deep, hierarchical neural networks with latent representations are increasingly widely and successfully applied to language tasks. As compositions of differentiable building blocks, deep models conventionally perform smooth, soft computations, resulting in dense hidden representations. In this work, we focus on models with **structure** and **sparsity** in both their **outputs** as well as their **latent representations**, without sacrificing differentiability for end-to-end gradient-based training. We develop methods for sparse and structured attention mechanisms, for differentiable sparse structure inference, for latent neural network structure, and for sparse structured output prediction. We find our methods to be empirically useful on a wide range of applications including sentiment analysis, natural language inference, neural machine translation, sentence compression, and argument mining.

BIOGRAPHICAL SKETCH

Vlad Niculae was born in Bucharest, Romania. He obtained a Bachelor's degree and a Master's degree in Computer Science from the University of Bucharest, where he developed an interest for natural language processing. He then pursued the Erasmus Mundus International Masters in Natural Language Processing and Human Language Technologies, spending one year at the University of Wolverhampton, and one year at the University of the Franche-Comté in Besançon.

Vlad is an accomplished open-source developer, a core contributor to the `scikit-learn` machine learning library, and lead developer of the `polylearn` library for factorization machines and polynomial networks, in addition to the open-source research code for the projects he has lead throughout his Ph.D. Vlad also enjoys movies, especially those depicting the everyday, and sometimes plays *Belle & Sebastian* songs on guitar.

Dedicated to my parents, Hermina & Marian, and to my sister Ana.

ACKNOWLEDGEMENTS

I am infinitely grateful to Claire, my advisor, for all of the help, support, patience, and ideas. I am equally thankful for my amazing minor advisors, Sabine, and Karthik. Part of this work was done during an internship at NTT with Mathieu, who gave me the courage to venture into new territories; many results later came from collaboration with André. I am happy without bounds to have become, from a big fan of your work, to collaborator. A proto-proof of Theorem 3.4 came up during a dinner conversation with Noelle. Uncountable thanks to Liviu, who introduced me to NLP research. Throughout the years, I have been blessed with fantastic friends, collaborators, and co-authors, namely:

Ithaca. Ana, Andreas, Andrew, Arzoo, Basu, Chenhao, Cristian, Danlu, Eleanor, Esin, Ethan, Felix, Isaac, Jack, Jon, Joshua, Justine, Kai, Laure, Lillian, Mark, Matt, Moontae, Natacha, Ozan, Rahmtin, Rishi, Sander, Shannon, Steffen, Sydney, Tianze, Tobias, Xanda, Xilun, Yao, Yiqing;

Nara. Antoine, David, Iwata-san, Juan, Katka, Michael, Otsuka-san, Ueda-san;

Saarbrücken. Anna, Beta, Bilal, Bimal, Ernesto, Eslam, Filip, Flor, Georg, Ghazale, Janno, Maëva, Marcos, Pedro, Przemek, Rafaella, Raul, Scott, Tahleen, Tine;

Elsewhere. Alex, Alexandra, Alina, Alison, Anca, Andrei, Anubhav, Bo, Bob, Calvin, Daniel, Emanuela, Fabian, Farid, Gaël, Gabriela, Iuliana, Jordan, Kelly, José, Lena, Maria, Miguel, Oana, Oda, Olivier, Răzvan, Radu, Raluca, Sashko, Sever, Silvia, Srijan, Tim, Vedrana, Weronika, Wilker, among others. I am also grateful to open-source developers and graduate student unions worldwide.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Vectors, matrices, and indexing.

$u, \boldsymbol{v}, \boldsymbol{W}$ a scalar, a vector, and a matrix, respectively;

v_i the ith element of vector $\boldsymbol{v}$;

w_{ij} the element on the ith row and jth column of $\boldsymbol{W}$

$\boldsymbol{w}_j$ the jth column of matrix $\boldsymbol{W}$;

$\boldsymbol{v}_{\mathcal{S}}$ the sub-vector of $\boldsymbol{v}$ indexed by an ordered set $\mathcal{S} \subset \mathbb{N}$;

$\boldsymbol{W}_{\mathcal{S}}$ the sub-matrix of $\boldsymbol{W}$ consisting of the columns indexed by $\mathcal{S}$;

$[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]$ vector concatenation, *i. e.*, for $\boldsymbol{v}_i \in \mathbb{R}^{d_i}$, $[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k] \in \mathbb{R}^{\sum_i d_i}$;

$[\boldsymbol{W}_1, \ldots, \boldsymbol{W}_k]$ column-wise stacking of $\boldsymbol{W}_i \in \mathbb{R}^{m \times d_i}$; $[\boldsymbol{W}_1, \ldots, \boldsymbol{W}_k] \in \mathbb{R}^{m \times \sum_i d_i}$;

$[\![n]\!]$ the index set, *i. e.*, $\{1, 2, \cdots, n\}$ for $n \in \mathbb{N}$.

Important sets and functions.

$\imath[\pi]$ the Iverson bracket: $\imath[\pi] = 1$ if proposition π is true, 0 otherwise;

$\|\boldsymbol{v}\|_p$ the ℓ_p norm of vector $\boldsymbol{v} \in \mathbb{R}^d$, *i. e.*, $\|\boldsymbol{v}\|_p = \left(\sum_{i=1}^d |v_i|^p\right)^{1/p}$;

$\triangle^d$ the canonical simplex, *i. e.*, $\{\boldsymbol{y} \in \mathbb{R}^d, \|\boldsymbol{y}\|_1 = 1, \boldsymbol{y} \geqslant \boldsymbol{0}\}$;

$\mathsf{Id}_{\mathcal{S}}$ the indicator function of set $\mathcal{S}$, $\mathsf{Id}_{\mathcal{S}}(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{S}$, else $+\infty$;

$\mathbf{P}_{\mathcal{S}}(\boldsymbol{v})$ euclidean projection of $\boldsymbol{v}$ onto the set S, *i. e.*, $\operatorname*{arg\,min}_{\boldsymbol{y} \in S} \|\boldsymbol{v} - \boldsymbol{y}\|_2^2$;

$\bar{\mathbb{R}}$ extended real numbers, *i. e.*, $\mathbb{R} \cup \{+\infty\}$

$\mathsf{dom}\, f$ the domain of function $f : \mathbb{R}^d \to \bar{\mathbb{R}}$, *i. e.*, $\left\{\boldsymbol{x} \in \mathbb{R}^d : f(\boldsymbol{x}) < \infty\right\}$;

$\partial f(\boldsymbol{x})$ the subdifferential of convex function $f : \mathbb{R}^d \to \bar{\mathbb{R}}$ at point $\boldsymbol{x}$, *i. e.*, $\partial f(\boldsymbol{x}) = \left\{\boldsymbol{g} \in \mathbb{R}^d : \forall \boldsymbol{z} \in \mathsf{dom}\, f, f(\boldsymbol{z}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^\top(\boldsymbol{z} - \boldsymbol{x})\right\}$; its elements $\boldsymbol{g} \in \partial f(\boldsymbol{x})$ are called *subgradients* of f at $\boldsymbol{x}$;

$\nabla f(\boldsymbol{x})$ the gradient of differentiable f at $\boldsymbol{x}$, *i. e.*, $\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$;

$\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})$ the Jacobian of $\boldsymbol{h} : \mathbb{R}^d \to \mathbb{R}^m$, *i. e.*, $\left(\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{x})\right)_{ij} = \frac{\partial h(\boldsymbol{x})_i}{\partial x_j}$;

$\boldsymbol{H}_f(\boldsymbol{x})$ the Hessian of $f : \mathbb{R}^d \to \mathbb{R}$, *i. e.*, $\left(\boldsymbol{H}_f(\boldsymbol{x})\right)_{ij} = \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i x_j}$;

CHAPTER 1

INTRODUCTION

In this chapter, we set the stage for presenting our contributions in context, we outline our work, and describe the organization of the remaining chapters.

1.1 Structure in Natural Language Processing

Computational methods are increasingly common for tackling challenging natural language processing (NLP) tasks. Machine learning approaches, in particular, are gaining success at a variety of challenging natural language problems. A popular example is *machine translation*, which takes as input a sentence in a source language, and aims to produce as output a translated sentence in the target language. Machine translation is deployed today at a large scale in successful commercial products.

Among the wide variety of computational methods considered for NLP applications, this thesis is focused on methods that identify and extract linguistic *structure*, in other words, discrete combinatorial representations of a text, reflecting the underlying linguistic phenomena at work. For decades, linguists have studied the tangled network of structural representations, manifesting at different scales. For example, through syntactic analysis, a sentence can be organized as a tree of *constituent* chunks, as formalized by Chomsky (1956); alternatively, *dependency* analysis yields a different kind of tree representation, where each word is a node, and arcs represent direct relations of grammatical dependency, a view deriving from the work of Tesnière (1959). At the document level, the perspective shifts to larger-scale structures, for example coreference, discourse, and argumentation. Between multiple texts, we may be interested in *alignment* structures (Harris, 1988).

For machine learning systems applied to NLP tasks, the input typically consists of text, while the output depends on the task of interest. Furthermore, many so-called *deep* models involve hidden (latent) representations computed along the way. A bird's eye view of a simple deep model for predicting the sentiment of a sentence is depicted in Figure 1.1a; the hidden representation depicted is, for instance, the dense vector output of a hidden layer inside a neural network.

Linguistic structure may play three main roles within a machine learning model:

- **Structured input.** In some circumstances, the input can be given with structural annotations; for instance, if users are required to input their text in a structured web-form, or if expert human annotators are employed. Structured input data is typically handled in a preprocessing or *feature extraction* step, and recent work employs neural models for graph inputs (Bruna et al., 2014; Beck et al., 2018, among others). While they represent a promising research direction, structured *input* models are out of the scope of our work.
- **Structured output** (Figure 1.1b). Instead of picking a category from a short list, the desired output itself might be a structured representation; for example, the most likely dependency parse tree of the given sentence. Structured output prediction (Bakır et al., 2007), especially in NLP (Smith, 2011), is characterized by highly expressive models, able to handle constraints and correlations at both a local level (for instance, which tag assignments are preferred or allowed for a given word) as well as at a global level (for instance, certain joint assignments may be disallowed). As such, finding the highest-scoring structure can be technically challenging.
- **Latent structure** (Figure 1.1c). In deep models, even if the desired output is unstructured, extracting **structured hidden representations** can poten-

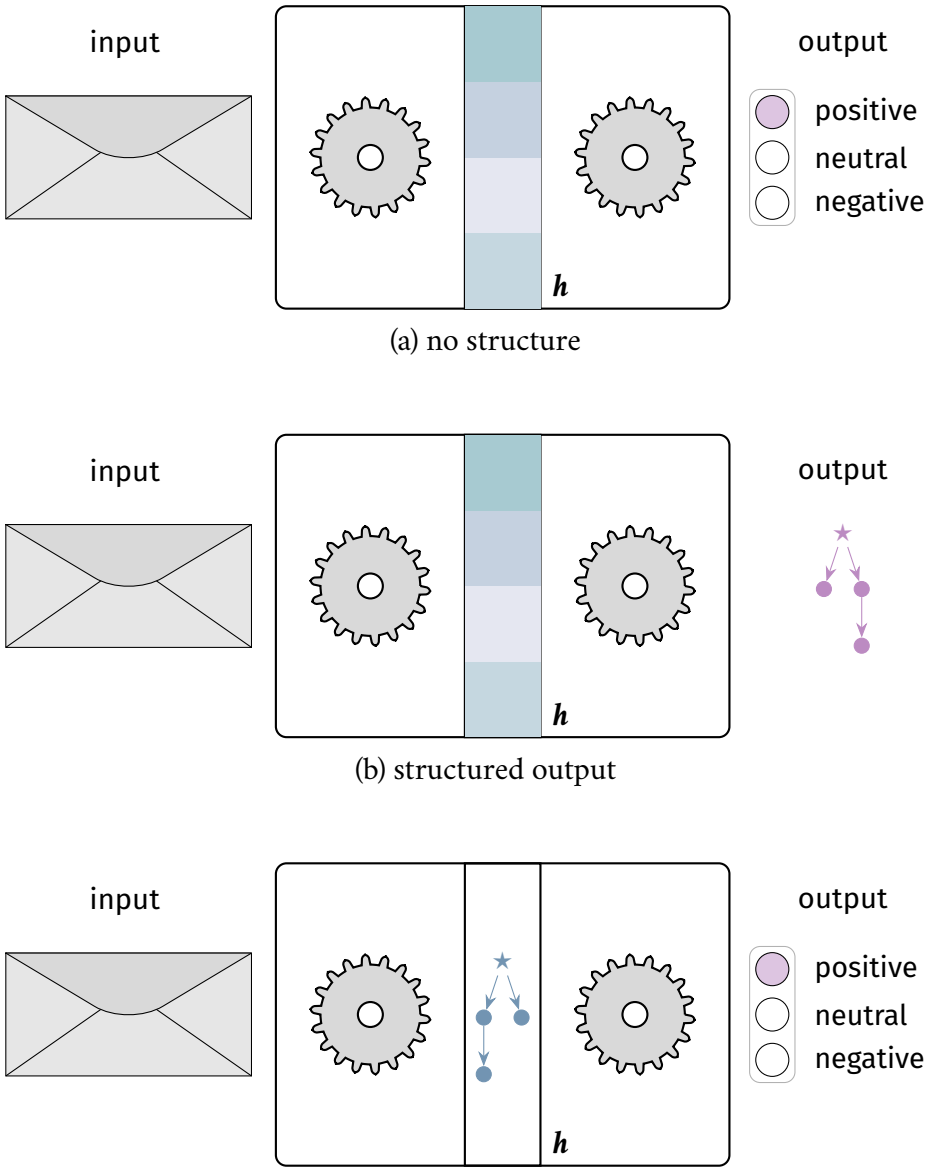


Figure 1.1: High-level view of *deep* machine learning models for NLP, with a hidden representation $\boldsymbol{h}$ emphasized: (a) a vanilla sentiment classifier, with unstructured hidden layers (typically, vector representations consisting of real numbers); (b) a structured-output model (*e. g.*, a parser); (c) a deep model with latent structure.

tially be beneficial for the downstream task; for example, taking a guess at the dependency structure of a sentence can help deal with scoped linguistic phenomena such as negation, leading to more accurate sentiment predictions. Latent structure inherits all the challenges of structured prediction

while facing additional ones, essentially due to the tension between discrete choices and gradient backpropagation, as we shall discuss in more detail in Chapter 3. Mitigating this tension through *sparsity* is a running theme in this dissertation.

In this work, we explore **structured output prediction** and **latent structure**, pushing the boundaries of model expressiveness, performance, and generality. We apply our approaches to a wide range of tasks, including machine translation, dependency parsing, natural language inference, and argument mining.

1.2 Contributions

Structured and sparse attention mechanisms via regularization. Neural attention is a recently developed mechanism for assigning latent probability weights to items (often, words within a sentence). We uncover a new perspective that casts attention mechanisms in terms of regularized *max* operators, leading to new derivations of well-known unstructured attention mechanisms. By drawing from extensive research on structured sparsity, our framework allows us to derive new attention mappings, which may encode structural priors. For example, in many languages, coherent phrases consist of adjacent words; thus we develop **fusedmax**: a linguistically-motivated attention mechanism tending to *group adjacent words together*. Since in some languages word order is variable, we also develop **oscarmax**, a mechanism that may *cluster non-adjacent words* as well. We showcase our proposed methods on sentence summarization, machine translation, and natural language inference, yielding superior interpretability with competitive performance and computational cost compared to traditional unstructured dense attention.

Differentiable sparse structured inference. For more complicated globally constrained structures, such as *matchings* or *dependency trees*, we turn to the framework of structured inference in probabilistic graphical models (Wainwright and Jordan, 2008). In particular, to tackle the challenge of searching over the enormous number of possible structures, we introduce SparseMAP, a new inference strategy. SparseMAP inference is able to automatically select only a few global structures: it is situated between *maximum a posteriori* (MAP) inference, which picks a single structure, and marginal inference, which assigns probability mass to all structures, including implausible ones. Importantly, SparseMAP can be computed using only calls to a MAP oracle, hence it is applicable even to problems where marginal inference is intractable, such as the linear assignment (or matching) problem. Sparsity makes gradient backpropagation efficient regardless of the structure, enabling us to augment deep neural networks with generic and sparse **structured hidden layers**. Experiments in natural language inference reveal competitive accuracy and improved interpretability when compared to unstructured mechanisms.

Latent neural network structure. Deep NLP models benefit from adapting their computation to underlying structures in the data; *e.g.*, TreeLSTMs using syntax as a hierarchical composition order. Yet, the structure is typically extracted using off-the-shelf parsers. Recent attempts to jointly learn the latent structure encounter a trade-off: they can either make factorization assumptions that limit expressiveness, or sacrifice end-to-end differentiability. Using our novel SparseMAP inference, which retrieves a sparse distribution over latent structures, we propose a novel approach for end-to-end learning of latent structure predictors jointly with a downstream predictor. Our method enables unrestricted dynamic computation graph construction from the *global* latent structure, while maintaining differentia-

bility. This approach leads to improved performance on tasks such as sentiment classification, natural language inference, and reverse dictionary lookup.

Structured Fenchel-Young Losses and `SparseMAP` losses. We derive a family of structured losses encompassing the conditional random field (CRF), the structured perceptron, and the structured SVM losses. We analyze some useful properties of this family, and use it to derive novel losses based on `SparseMAP` inference. Exploiting the sparse distribution over structures produced by `SparseMAP` is valuable in practice, as we demonstrate on dependency parsing, where we outperform the aforementioned losses on most languages considered. Our parsers get increasingly sparser as training progresses, peaking on a single tree for unambiguous sentences. On sentences with inherent linguistic ambiguity, `SparseMAP` parsers retrieve a small set of candidate parse trees, helping both practitioners and downstream applications in pipeline systems.

Expressive neural structured models for argument mining. To validate the importance of incorporating structure and domain knowledge in specialized NLP applications, we study in greater detail one specific application, *argument mining*, whose goal is to extract argumentation structures from documents. We construct an expressive graphical model, capturing the domain knowledge present in two different but related argumentation datasets. Structured output prediction techniques enable our model to jointly learn elementary unit type classification and argumentative relation prediction, capturing correlations between adjacent relations as well as global constraints. Experimental results reveal that global structured models are essential for argument mining, outperforming unstructured baselines.

1.3 Roadmap

The remainder of this thesis is organized as follows.

We begin by reviewing, in Chapter 2, the relevant background in neural network models for NLP, convex analysis, and structured sparsity.

Chapters 3, 4, and 5 study **latent structure** in neural networks. In Chapter 3, we develop a generalized framework for *neural attention mechanisms* based on regularization, allowing us to develop differentiable attention mechanisms that enforce structured sparsity. In Chapter 4 we propose `SparseMAP`, a novel strategy for differentiable sparse structured inference, with efficient algorithms for computing the forward and backward passes provided access to MAP inference. Next, in Chapter 5, we develop a method for using `SparseMAP` to train neural networks whose computation graphs depend freely and directly on structured latent variables.

Chapters 6 and 7 are concerned with **structured output prediction**. In Chapter 6 we introduce a family of structured losses generalizing the most commonly used ones. We propose `SparseMAP` losses as members of this family and explore their theoretical and practical properties and performance. In Chapter 7 we design and evaluate a powerful structured output model for argumentation mining, reaffirming the importance of incorporating structure and domain knowledge in NLP models.

We conclude in Chapter 8 by contextualizing our contributions and the future work envisioned in the current landscape of structured models for NLP.

CHAPTER 2

BACKGROUND

This chapter serves as a reminder of the topics our work builds upon: machine learning, neural network architecture, convex analysis, and structured sparsity.

2.1 Machine Learning with Hidden Representations of Language

We begin by providing a more rigorous and practical explanation of the machine learning pipeline for NLP, shown in Figure 1.1.

Given an input prompt x from a set of possible inputs $\mathcal{X}$ (for instance, the set of all possible English sentences), we want to find the most likely output $y \in \mathcal{Y}$. Various examples of objects we might be interested in predicting include

- the sentence's sentiment: $\mathcal{Y} = \{\mathsf{negative}, \mathsf{neutral}, \mathsf{positive}\}$
- the writer's age: $\mathcal{Y} = \mathbb{R}_+$
- the *dependency tree* between the words in a sentence: $\mathcal{Y} = \mathcal{T}$, *i. e.*, the set of all directed trees with a single root.

We formalize this via a scoring function which, given x, should assign higher scores to the correct output y than to incorrect ones

$$\theta : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}. \tag{2.1}$$

We can make predictions by selecting the highest-scoring output

$$\hat{y}(x) = \operatorname*{arg\,max}_{y \in \mathcal{Y}} \theta(y; x),$$

and, where obvious from context, we may drop the argument and denote the prediction simply as $\hat{y}$.

Supervised learning. Instead of designing the score θ entirely by hand, machine learning entails learning a good model for θ based on *training data*. We thus assume access to a set of N training inputs paired with desired outputs

$$\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}|_{i=1}^{N}.$$

To facilitate the exploration of the space of scoring functions σ, let us consider *parametrized* functions $\theta_{\boldsymbol{w}}$, where we denote by $\boldsymbol{w}$ the parameter weights. Then, we may pose the learning problem as finding $\boldsymbol{w}$ such that $\theta_{\boldsymbol{w}}$ matches the training data well. Ideally, we would want to minimize the number of incorrect predictions

$$\underset{\boldsymbol{w}}{\text{minimize}} \sum_{i=1}^{N} \imath[\hat{y}(x_i) \neq y_i], \tag{2.2}$$

with $\hat{y}$ depending on $\theta_{\boldsymbol{w}}$ as in Equation 2.1. This discrete optimization problem is typically not tractable, so a more amenable surrogate to the zero-one error is considered. Denoting by $\boldsymbol{\theta}_{\boldsymbol{w}}(x) \in \mathbb{R}^{|\mathcal{Y}|}$ the vector obtained by applying the scoring function to every possible output y, we want to minimize

$$\underset{\boldsymbol{w}}{\text{minimize}} \sum_{i=1}^{N} L(\boldsymbol{\theta}_{\boldsymbol{w}}(x), y_i). \tag{2.3}$$

The key ingredient in Equation 2.3 is the **loss function** L, which measures the discrepancy between the scores and the true object. Choosing a suitable loss function depends on many factors, including the mathematical properties of the function, as well as the type of target objects y; this subject will be discussed in more detail in Chapter 6. One example is the *hinge loss* for classification

$$L(\boldsymbol{\theta}(x), y) = \max\left(0, 1 + \max_{y' \neq y} \theta(y'; x) - \theta(y; x)\right). \tag{2.4}$$

Linear models. A simple way to define the function $\sigma_{\boldsymbol{w}}$ is as a linear function of the input. This view requires a numeric representation of the data as vectors of features. Usually, data is not provided directly in vector form, so practitioners must employ **feature extraction**. We may represent feature extraction as a function $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^d$. Then, a linear model for classification takes the form

$$\theta_{\boldsymbol{w}}(y; x) = \boldsymbol{w}_y^\top \boldsymbol{\phi}(x),$$

where the weight vector is re-organized as the concatenation of per-class weights, *i. e.*, $\boldsymbol{w} = [\boldsymbol{w}_y]\big|_{y \in \mathcal{Y}}$. Linear models are appealing because of their simplicity, and the ease of optimizing $\boldsymbol{w}$ in many situations.

Typically, the feature extractor g is defined by hand, by implementing it programatically. For example, if x is a sentence, the first feature $\big(\phi(x)\big)_1$ may be defined as the number of words in x, and the second feature $\big(\phi(x)\big)_2$ may be defined as 1 if the last character is a question mark, and 0 otherwise. Practitioners dedicate plenty of effort to finding good feature representations, in order to improve predictive performance—an endeavor commonly known as **feature engineering**.

Deep models. Deep learning is a highly successful alternative to feature engineering, where σ can be represented as an arbitrary composition of *hidden layers*. To showcase why this is useful, considering replacing the feature extractor $\boldsymbol{\phi}$ with a learnable function $\boldsymbol{\phi}_{\boldsymbol{w}}$, with the goal of learning appropriate feature representations instead of having to engineer them manually. We may write

$$\theta_{\boldsymbol{w}}(y; x) = \theta'_{\boldsymbol{w}}(y; \boldsymbol{h}) \quad \text{where} \quad \boldsymbol{h} = \boldsymbol{\phi}_{\boldsymbol{w}}(x),$$

thereby identifying $\boldsymbol{h} \in \mathbb{R}^d$ as a *hidden* representation of the input. The remainder of the model θ' may be recursively defined through more hidden layers, but deep

models are not limited to *sequential* composition: an arbitrary *computation graph* may be used to describe the operations to be performed. For instance, the input may be fed directly into deeper layers (so-called *skip connections*), and weights may be shared between layers (leading to convolutional and recurrent networks, among others). The computation graph abstraction is extremely powerful, due to the resulting flexibility and modularity (Goodfellow et al., 2016).

Backpropagation. Unlike linear models, for which learning, *i. e.*, optimizing Equation 2.3, is relatively simple, parameter learning in deep models can be more difficult. A popular approach is stochastic *gradient-based* optimization. These algorithms perform well empirically and make minimal assumptions about the model θ_w: all that is needed is a way to compute the gradient of the loss with respect to the model weights:

$$\frac{\partial L(\theta_w(x), y)}{\partial w}.$$

Backpropagation is an algorithm for evaluating this gradient at a given point, provided access to *Jacobian-vector* products at each computation node in the graph (Nocedal and Wright, 1999, Chapter 8.2). This allows researchers to develop neural network modules as **building blocks** that practitioners may compose together in new and creative ways: a programming paradigm that has proven very fruitful.

2.2 Convex Analysis

In this section, we recapitulate some useful definitions and results from the theory of convex functions and sets, a crucial foundation for our results.

Definitions. A set $\mathcal{S}$ is called a **convex set** if it contains any segment whose endpoints are in $\mathcal{S}$, in other words, if the following holds

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{S}, \forall \alpha \in [0,1], \qquad \alpha \boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2 \in \mathcal{S}.$$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is called a **convex function** if $\mathsf{dom}\, f$ is convex and the following property holds

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathsf{dom}\, f, \forall \alpha \in [0,1], \qquad f(\alpha \boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2) \leq \alpha f(\boldsymbol{x}_1) + (1-\alpha) f(\boldsymbol{x}_2).$$

In the above, the pair $(\alpha, 1-\alpha)$ are called coefficients of a **convex combination**. More generally, a convex combination of k points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k$ is the weighted average given by coefficients $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_k] \in \triangle^k$ as

$$\sum_i \alpha_i \boldsymbol{x}_i = \boldsymbol{X}\boldsymbol{\alpha},$$

where $\boldsymbol{X}$ is a matrix whose columns are the n points, *i. e.*, $\boldsymbol{X} = [\boldsymbol{x}_1^\top, \cdots, \boldsymbol{x}_k^\top]$.

Convex hulls and polytopes. The convex hull of a set $\mathcal{S}$ is defined as the set of all convex combinations of points in $\mathcal{S}$, *i. e.*,

$$\mathsf{conv}\, \mathcal{S} = \left\{ \sum_{i=1}^{k} \alpha_k \boldsymbol{x}_k : \boldsymbol{\alpha} \in \triangle^k, \boldsymbol{x}_i \in \mathcal{S}\ \forall i \in [\![k]\!] \right\}.$$

We use the term **polytope** to denote the convex hull of a finite set of points. Given a polytope $\mathcal{P}$, there is a unique minimal set of points $\mathcal{V}$ such that $\mathcal{P} = \mathsf{conv}\, \mathcal{V}$, (minimal in the sense that $\forall \mathcal{V}' \subsetneq \mathcal{V}\ \ \mathsf{conv}\, \mathcal{V}' \neq \mathcal{P}$). The elements of $\mathcal{V}$ are called the **vertices** of $\mathcal{P}$. The $k-1$-dimensional **canonical (probability) simplex** $\triangle^k$ is the polytope with the k basis vectors $\boldsymbol{e}_1, \cdots, \boldsymbol{e}_k$ as vertices. It follows that, any k-vertex polytope is the image of the $\triangle^k$ through the linear mapping given by $\boldsymbol{V}$

$$\mathcal{P} = \{\boldsymbol{V}\boldsymbol{\alpha} : \alpha \in \triangle^k\},$$

where the columns $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k$ of $\boldsymbol{V}$ are the vertices of $\mathcal{P}$. Points in $\mathcal{P}$ where at least one coordinate of $\boldsymbol{\alpha}$ is zero form the *relative boundary* of $\mathcal{P}$, while all others form its *relative interior*:

$$\operatorname{relint} \mathcal{P} = \{\boldsymbol{V}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \triangle^k, \boldsymbol{\alpha} > \mathbf{0}\}. \tag{2.5}$$

Any polytope can also be characterized as the solution set of a system of linear equalities and inequalities; this view proves helpful when explicitly writing optimality conditions of constrained optimization problems.

The convex conjugate. Given a function $f : \mathbb{R}^d \to \bar{\mathbb{R}}$, we define its conjugate, also known as its *Legendre-Fenchel transformation*, as

$$f^* : \mathbb{R}^d \to \mathbb{R} \qquad f^*(\boldsymbol{y}) = \sup_{\boldsymbol{x} \in \operatorname{dom} f} \boldsymbol{y}^\top \boldsymbol{x} - f(\boldsymbol{x}). \tag{2.6}$$

The convex conjugate of a function is convex even when f is not. An important property of convex conjugates is the Fenchel-Young inequality (Fenchel, 1949)

$$f(\boldsymbol{x}) + f^*(\boldsymbol{y}) \geq \boldsymbol{x}^\top \boldsymbol{y}. \tag{2.7}$$

Boyd and Vandenberghe (2004, Section 3.3.2) provide more information and properties of the convex conjugate.

Proximal operators. Given a convex function $f : \mathbb{R}^d \to \mathbb{R}$, its proximal operator is defined as (Parikh and Boyd, 2014)

$$\mathbf{prox}_f : \mathbb{R}^d \to \mathbb{R}^d \qquad \mathbf{prox}_f(\boldsymbol{v}) = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \tfrac{1}{2} \|\boldsymbol{x} - \boldsymbol{v}\|_2^2 \tag{2.8}$$

Under mild assumptions on f, this $\arg\min$ is unique. In particular, the proximal operator of the identity function of a convex set $\mathcal{S} \subset \mathbb{R}^d$ is simply the euclidean

projection onto $\mathcal{S}$

$$\mathbf{prox}_{\mathsf{Id}_{\mathcal{S}}}(\boldsymbol{v}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^d} \mathsf{Id}_{\mathcal{S}}(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{v}\|_2^2 \quad = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{S}} \|\boldsymbol{x}-\boldsymbol{v}\|_2^2 \,. \tag{2.9}$$

This suggests an interpretation of proximal operators as *generalized projections*.

2.3 Sparsity, Structured Sparsity, and Parsimony

The principle of **parsimony** states that, all things being equal, a simple model should be preferred to a more complex one. Simplicity can be defined in many ways, but generally, simple models are understood to be:

- more computationally **efficient**, for instance via compact representations that require less storage;
- easier to **interpret** and visualize, since sparse explanations require less cognitive effort to reason about;
- more **plausible** in the presence of uncertainty and noise: even when we don't know for sure if the true phenomenon is simple, it may be a good idea to "bet on simplicity".[1]

Sparsity is a typical measure of simplicity: a vector $\boldsymbol{x} \in \mathbb{R}^d$ is sparse if many of its coordinates are exactly zero. The number of non-zero coordinates of a vector is denoted as below, and can be seen as a complexity measure.

$$\|\boldsymbol{x}\|_0 = \big|\{i \in [\![d]\!] : x_i \neq 0\}\big| \tag{2.10}$$

[1] We are slightly paraphrasing the "bet on sparsity" as formulated by Hastie et al. (2015): "Use a procedure that does well in sparse problems, since no procedure does well in dense problems."

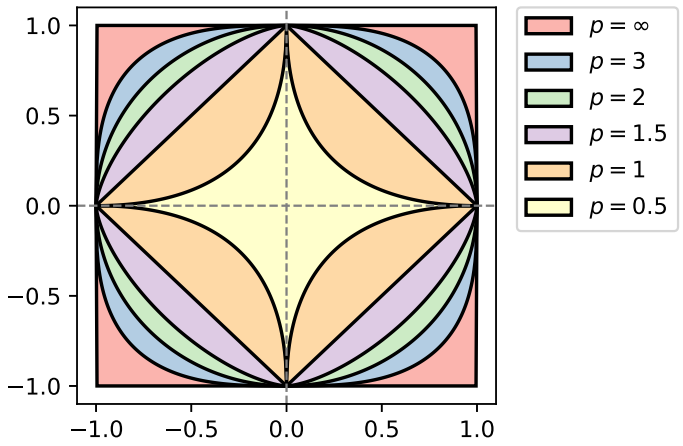


Figure 2.1: ℓ_p norm balls $\mathcal{B}_{\|\cdot\|_p}$ in a two-dimensional space, for several values of p. The legend matches the outside-to-inside ordering of the contours. For $p < 1$, ℓ_p are not proper norms; as $p \to 0$ the limit vectors have exactly one nonzero coordinate.

Often called the ℓ_0 norm by abuse of language, this function is not a metric norm, but, if the domain is bounded, it is a limit of ℓ_p norms, namely $\|\cdot\|_0 = \lim_{p\to 0} \|\cdot\|_p^p$. The ℓ_0 function is discontinuous and non-convex, and optimization problems involving it are typically NP-hard. However, other p-norms can be used to **induce sparsity** while leading to simpler optimization problems. The most commonly used such surrogate is the ℓ_1 norm, $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$. It is convex and continuous, and minimizing or constraining its value yields sparse solutions.

To see why, we introduce the **unit ball** of an ℓ_p penalty function

$$\mathcal{B}_{\|\cdot\|_p} = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\|_p \leq 1\}.$$

Figure 2.1 illustrates unit balls for several interesting norms. In particular, it can be seen that $\mathcal{B}_{\|\cdot\|_1}$ is a polytope with vertices $\{\pm\boldsymbol{e}_i : i \in [\![d]\!]\}$. The fundamental theorem of linear programming states that the minimum of a linear function over a polytope (including $\mathcal{B}_{\|\cdot\|_1}$) is always attained at a vertex (Dantzig et al., 1955, Theorem 6); similarly, the minimum of a quadratic function over a polytope is

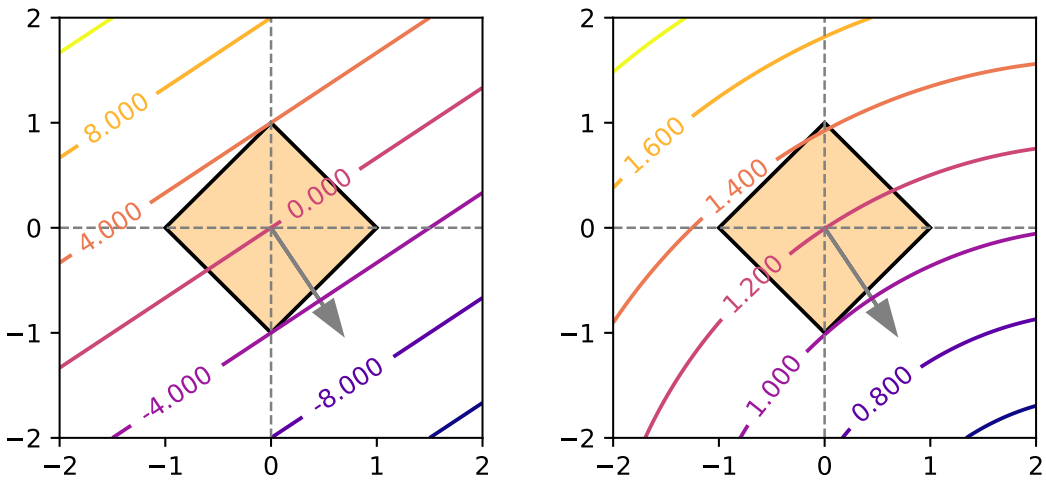


Figure 2.2: When optimizing a linear function (left) or a quadratic function (right) over the ℓ_1 ball, solutions are likely sparse.

likely (but not always) attained at a vertex. Both phenomena are illustrated in Figure 2.2. Since euclidean projections are quadratic minimizations, it follows that $\mathbf{P}_{\mathcal{B}_{\|\cdot\|_1}}$ has sparse solutions. We further note that the d-simplex is a face of the d-dimensional ℓ_1 ball, suggesting that similar optimization problems over the simplex also result in sparse solutions; we later prove a more general result about what probability mappings are sparsity-inducing in Proposition 3.1.

Vertex sparsity is a notion of simplicity for points in a polytope $\mathcal{P} = \operatorname{conv} \mathcal{V}$. Recall that any $\boldsymbol{x} \in \mathcal{P}$ can be represented as a convex combination of its vertices:

$$\forall \boldsymbol{x} \in \mathcal{P} \; \exists \, \boldsymbol{\alpha} \in \triangle^{|\mathcal{V}|}, \quad \boldsymbol{x} = \sum_{\boldsymbol{v} \in \mathcal{V}} \alpha_{\boldsymbol{v}} \boldsymbol{v}. \tag{2.11}$$

In some cases, for instance if all vertices are strictly positive vectors, *i. e.*, $\boldsymbol{v} \succ \mathbf{0}$, $\mathcal{P}$ may even contain no points with sparse *coordinates*. Yet, by shifting to the representation in Equation 2.11, we find that some points can be compactly represented as sparse *convex combinations* involving only a few vertices, *i. e.*, $\|\boldsymbol{\alpha}\|_0 \ll |\mathcal{V}|$. This can be just as desirable as coordinate sparsity. Moreover, if the vertices themselves

are sparse vectors, then vertex sparsity results in coordinate sparsity as well.[2]

Group sparsity. In some cases, the coordinates of a vector $\boldsymbol{x} \in \mathbb{R}^d$ have some known meaning, and we might want several coordinates to either all be zero or all be nonzero. We may organize the indices into groups $\mathcal{G}_i \subset [\![d]\!]_{i=1}^k$. The *group lasso* penalty (Yuan and Lin, 2006) is defined as

$$R_{\mathrm{GL}}(\boldsymbol{x}) = \sum_{i=1}^{k} \left\| \boldsymbol{x}_{\mathcal{G}_i} \right\|_2 .$$

If $k = d$ and each coordinate is in a separate group, *i. e.*, $\mathcal{G}_i = \{i\}$, then the group lasso reverts to the ℓ_1 penalty discussed above. While originally proposed for non-overlapping groups, the group lasso has been extended to unions and intersections of overlapping groups (Jacob et al., 2009; Jenatton et al., 2011).

Fusing values. Parsimony can go beyond zeroing out coordinates, vertices or groups. We may also encourage simplicity in a vector by clustering (fusing) together its *values*. An important instance of this is the **fused lasso**, used to smoothen or denoise a 1-d sequence (represented as a vector $\boldsymbol{x}$ by encouraging adjacent coefficients to be exactly equal. This can be done by penalizing the absolute value of their difference (Tibshirani et al., 2005)

$$R_{\mathrm{FL}}(\boldsymbol{x}) = \sum_{i=1}^{d-1} |x_{i+1} - x_i|. \tag{2.12}$$

More generally, given a graph over the d indices, weighted by a matrix $\boldsymbol{W} \in \mathbb{R}^{d \times d}$, we may define a *generalized* fused lasso penalty (Tibshirani and Taylor, 2011)

$$R_{\mathrm{GFL}}(\boldsymbol{x}) = \sum_{i<j} w_{ij} |x_i - x_j|. \tag{2.13}$$

[2]Note that when $\mathcal{P} = \triangle$, coordinate sparsity and vertex sparsity coincide.

Unlike the 1-d version, for which efficient exact algorithms exist, the general formulation is more challenging.

In some cases, the actual ordering of the coefficients doesn't matter, but we might still desire a parsimonious vector with only a few distinct values. The **OSCAR** regularizer (Bondell and Reich, 2008) achieves this effect without incurring the high computational cost of the generalized fused lasso.

$$R_{\mathrm{OSC}} = \lambda_1 \|\boldsymbol{x}\| + \lambda_2 \sum_{i<j} \max(|x_i|, |x_j|). \tag{2.14}$$

Generalized penalties. There has been a lot of interest in the study of more generalized structured norms and penalties. We briefly list some directions, useful not only in hand-crafting better suited penalties for a task as hand, but also for their analysis which can lead to insights into the penalties discussed above.

The **ordered weighted ℓ_1 (OWL)** norm (Zeng and Figueiredo, 2015) is

$$R_{\mathrm{OWL}}(\boldsymbol{x}) = \sum_i w_i |x_i^{\downarrow}|,$$

where $x_i^{\downarrow}$ denotes the ith largest element of $\boldsymbol{x}$. With the weight defined as $w_i = 1$, OWL is equal to the ℓ_1 norm; when $w_1 = 1$ and $w_i = 0$ for $i > 1$ it amounts to ℓ_∞, and for $w_i = \lambda_1 + \lambda_2(d - i)$, OWL becomes the OSCAR regularizer in Equation 2.14.

Another perspective is given through **atomic norms** (Chandrasekaran et al., 2012). A set of *atoms* $\mathcal{A}^k \subset \mathbb{R}^d$ induces, under some assumptions, the norm

$$\|\boldsymbol{x}\|_{\mathcal{A}} = \inf\{t \geq 0 : \boldsymbol{x} \in t \operatorname{conv} \mathcal{A}\}.$$

Atomic norms induce representations as sparse affine combinations of atoms. For $\mathcal{A} = \{\pm \boldsymbol{e}_i : i \in [\![d]\!]\}$, $\|\cdot\|_{\mathcal{A}} = \|\cdot\|_1$; for $\mathcal{A} = \{-1, 1\}^d$, $\|\cdot\|_{\mathcal{A}} = \|\cdot\|_\infty$. An atomic formulation of the OWL norm is given by Zeng and Figueiredo (2015).

CHAPTER 3

STRUCTURED SPARSITY FOR ATTENTION MECHANISMS

In this chapter, we focus on neural attention mechanisms: latent selector modules which decide on what part of the input a network should focus. We empower these mechanisms with **sparsity**, in particular structured sparsity, by proposing a regularization-based framework for attention. We recover within this framework the popular attention mechanisms known as softmax and sparsemax, leading to new insights into them.

Building on well-studied structured penalties, we develop new and more interpretable attention mechanisms, that focus on entire segments or groups of an input. We derive efficient algorithms to compute the forward and backward passes of our attention mechanisms, enabling their use within neural networks. Our attention mechanisms are efficient and interpretable drop-in replacements for softmax. For certain applications, the structured priors incorporated by our methods can lead to superior results.

This chapter is based on Niculae and Blondel (2017).

3.1 Motivating Structured Sparsity For Attention

Modern neural network architectures are commonly augmented with an attention mechanism, which tells the network where to look within the input in order to make the next prediction. Attention-augmented architectures have been successfully applied to machine translation (Bahdanau et al., 2015; Luong et al., 2015), speech recognition (Chorowski et al., 2015), image caption generation (Xu et al., 2015), textual entailment (Rocktäschel et al., 2016; Martins and Astudillo, 2016), and

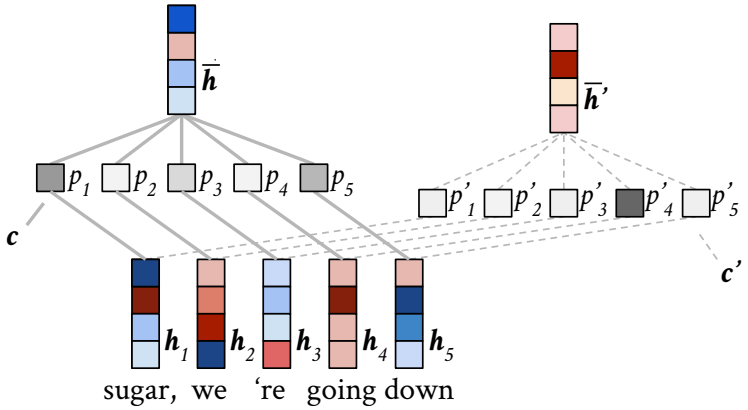


Figure 3.1: A generic attention mechanism computes a representative vector $\bar{\boldsymbol{h}}$ as a context-dependent weighted average of a sequence of vectors ($\boldsymbol{h}_i$), by inducing a probability distribution over them, *i. e.*, $\bar{\boldsymbol{h}} = \sum_i p_i \boldsymbol{h}_i$. For different contexts $\boldsymbol{c}$ and $\boldsymbol{c}'$, the distributions $\boldsymbol{p}$ and $\boldsymbol{p}'$ may be wildly different.

sentence summarization (Rush et al., 2015), to name but a few examples.

Attention is used to select, out of a variable-length list of vectors representing items (*e. g.*, words), a single representative vector, given some context (Figure 3.1). A popular and illustrative application is in *sequence to sequence* (*seq2seq*) neural machine translation, where a recurrent neural net (RNN) *encoder* first transforms each of the d source words into a vector $\boldsymbol{h}_i \in \mathbb{R}^k$, where $i \in [d]$. Then, the *decoder* incrementally predicts target words, using a traditional multi-class classifier (with one class per known word in the target language). The input to this classifier, however, is context-sensitive: at each time step t, the input combines the current *decoder hidden state context* $\boldsymbol{c}^{(t-1)}$, and the *attention vector* which approximates the most relevant source word, given the current context:

$$\bar{\boldsymbol{h}}^{(t)} = \sum_{i=1}^{d} p_i \boldsymbol{h}_i \quad \text{where} \quad p_i = p_i(\boldsymbol{h}_i, \boldsymbol{c}^{(t-1)}).$$

In an idealized simplified case, when translating one word, we would need to look only at the unique relevant source word j, so we would like the probabilities $\boldsymbol{p}$ to concentrate on the one-hot vector $\boldsymbol{e}_j$. Relaxing this discrete selection to

a continuous probability allows the model to hedge its bets and capture *uncertainty*. Importantly, continuous attention also allows such models to be trained via backpropagation, since the probabilities $\boldsymbol{p}$ are differentiable.

Estimating this latent relevance probability is not trivial. The key insight behind attention mechanisms is to parametrize this probability using two components:

1. a regression-like **scorer** module, generating a relevance score for each word $\boldsymbol{h}_i$ relative to some context $\boldsymbol{c}$:

$$(\boldsymbol{h}_i, \boldsymbol{c}) \rightarrow \theta_i \in \mathbb{R}$$

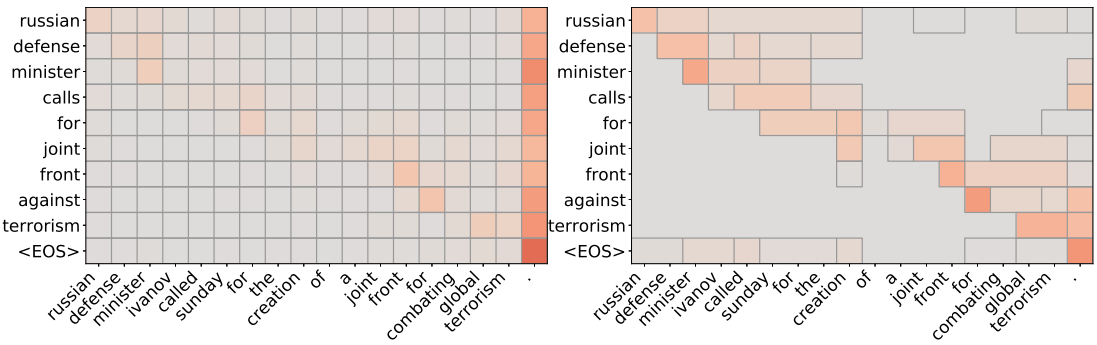
2. a normalizing **probability** mapping $\Pi : \theta \rightarrow \boldsymbol{p}$ from scores into probabilities. By far, the most common such mapping is the softmax:

$$p_i = \mathsf{softmax}(\boldsymbol{\theta})_i = \frac{\exp \theta_i}{\sum_{j=1}^{d} \exp \theta_j}$$

These two components are present in all newly-proposed types of attention mechanisms: self-attention (Lin et al., 2017), key-value attention (Daniluk et al., 2017), pointer networks (Vinyals et al., 2015), etc.

Alongside empirical successes, neural attention—while not necessarily correlated with human attention—is increasingly crucial in bringing more **interpretability** to neural networks by helping explain how individual input elements contribute to the model's decisions. It is common to inspect and report attention distribution as heatmap plots; for *seq2seq* models, such plots often look like Figure 3.2a.

A notable property of softmax is that its outputs are **always dense**: there are no scores $\boldsymbol{\theta}$ such that $\mathsf{softmax}(\boldsymbol{\theta})_k = 0$ for some k. For simplicity, human operators



(a) softmax (b) fusedmax

Figure 3.2: Traditionally, attention mechanisms use softmax for mapping scores to probabilities, yielding dense, not very interpretable results (3.2a). Within our framework, we can derive attention mechanisms that yield sparse, clustered probabilities, aiding interpretability (3.2b).

will focus only on the first few highest-weighted items, but because the attention weights are never zero, all elements in the input always make at least a small contribution to the decision. This leads to a disconnect between our *perception* of the model and the actual model.

To overcome this limitation, Martins and Astudillo (2016) recently proposed *sparsemax*, using the Euclidean projection onto the simplex as a sparse alternative to softmax. But, as we have seen in Section 2.3, the principle of parsimony, stating that simple explanations should be preferred over complex ones, goes well beyond such *coordinate-level* sparsity: depending on the application, it may be useful to consider selection of entire groups, of rectangles and convex shapes, or of equally-weighted *clusters*. Such properties, thoroughly studied in the field of *structured sparsity*, can lead to better interpretability, as well as to more adequate *structural prior knowledge* assumed by the model.

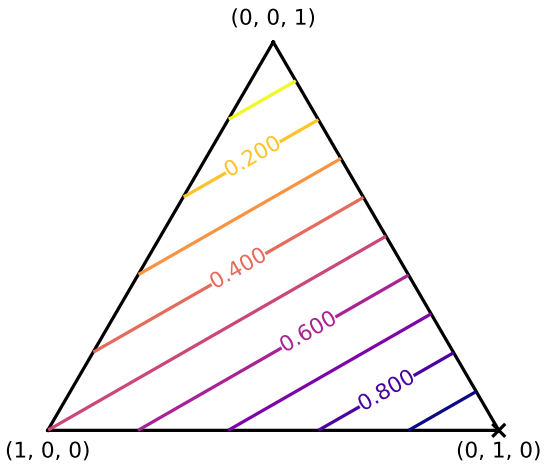


Figure 3.3: Finding the maximum coordinate of a vector (here, $\theta = [.5, 1, 0]$) is equivalent to maximizing a linear function over the simplex. As the simplex is non-empty, a solution is always achieved at a vertex.

3.2 Regularized max Operators

In this section, we shed light on the intimate connection between the max function and an arg max-like mapping from $\mathbb{R}^d$ to Δ^d. Using a smoothing technique from convex analysis, we extend this intuition to an entire family of regularized max operators and their probability mappings, recovering well-known expressions.

Reformulating max as an optimization problem. The maximum operator is a function from $\mathbb{R}^d$ to $\mathbb{R}$ and can be defined by

$$\max(\theta) \coloneqq \max_{i \in [d]} \theta_i = \sup_{\boldsymbol{p} \in \Delta^d} \boldsymbol{p}^\top \theta. \tag{3.1}$$

The equality on the right-hand is an essential insight, and it stems from the fact that the supremum of a linear form over the simplex is always achieved at a vertex. This is a direct consequence of the fundamental theorem of linear programming (Dantzig et al., 1955, Theorem 6), illustrated in Figure 3.3. Moreover, by Danskin's theorem (Danskin, 1966), any optimal $\boldsymbol{p}^\star$ is a subgradient of the supremum. More

strongly, we have

$$\partial \max(\boldsymbol{\theta}) = \operatorname{conv}\{\boldsymbol{e}_{i^\star} : i^\star \in \arg\max_{i \in [d]} \theta_i\}. \tag{3.2}$$

We can see $\partial \max(\boldsymbol{\theta})$ as a mapping $\Pi \colon \mathbb{R}^d \to \triangle^d$. When there are no ties, $\partial \max$ concentrates all probability mass onto the highest-scoring item: $\Pi(\theta) = \boldsymbol{e}_{i^\star}$. This mapping is, however, ill-behaved in crucial ways: it is multi-valued whenever there are ties, it is discontinuous, and it is piecewise constant wherever continuous (since small enough changes to $\boldsymbol{\theta}$ do not change the maximum, in general). Therefore, $\partial \max$ is not amenable to optimization by gradient descent, and thus unsuitable for direct use in neural network hidden layers.

A regularized max operator and its gradient mapping. These shortcomings encourage us to consider a regularization of the maximum operator. Inspired by the seminal work of Nesterov (2005), we apply a smoothing technique. The conjugate of $\max(\boldsymbol{\theta})$ is

$$\max^*(\boldsymbol{p}) = \mathsf{Id}_{\triangle^d} = \begin{cases} 0, & \text{if } \boldsymbol{p} \in \triangle^d \\ \infty, & \text{otherwise.} \end{cases} \tag{3.3}$$

To prove this, we note that $\mathsf{Id}^*_{\triangle^d} = \max$ (Boyd and Vandenberghe, 2004, Example 3.24), and that, since $\triangle^d$ is closed and convex, $\mathsf{Id}^{**}_{\triangle^d} = \mathsf{Id}_{\triangle^d}$. Taking the conjugate of both sides leads to the desired result. We proceed to add regularization to $\max^*$

$$\max^*_\Omega(\boldsymbol{p}) \coloneqq \mathsf{Id}_{\triangle^d} + \gamma\Omega(\boldsymbol{p}) = \begin{cases} \gamma\Omega(\boldsymbol{p}), & \text{if } \boldsymbol{p} \in \triangle^d \\ \infty, & \text{otherwise.} \end{cases} \tag{3.4}$$

where we assume that $\Omega \colon \mathbb{R}^d \to \mathbb{R}$ is β-**strongly convex** w.r.t. some norm $\|\cdot\|$ and $\gamma > 0$ controls the regularization strength. To define a smoothed $\max$ operator, we

take the conjugate once again

$$\max_\Omega(\boldsymbol{p}) \coloneqq \max_\Omega^{**}(\boldsymbol{\theta}) = \sup_{\boldsymbol{p}\in\mathbb{R}^d} \boldsymbol{p}^\top\boldsymbol{\theta} - \max_\Omega^*(\boldsymbol{p}) = \sup_{\boldsymbol{p}\in\triangle^d} \boldsymbol{p}^\top\boldsymbol{\theta} - \gamma\Omega(\boldsymbol{p}) \tag{3.5}$$

Our main proposal is a mapping $\Pi_\Omega\colon \mathbb{R}^d \to \triangle^d$, defined as the *argument* that achieves this supremum.

$$\boxed{\Pi_\Omega(\boldsymbol{\theta}) \coloneqq \arg\max_{\boldsymbol{p}\in\triangle^d} \boldsymbol{p}^\top\boldsymbol{\theta} - \gamma\Omega(\boldsymbol{p}) = \nabla\max_\Omega(\boldsymbol{\theta})} \tag{3.6}$$

The right-hand side follow from i) $\max_\Omega(\boldsymbol{\theta}) = (\boldsymbol{p}^\star)^\top\boldsymbol{\theta} - \max_\Omega^*(\boldsymbol{p}^\star) \Leftrightarrow \boldsymbol{p}^\star \in \partial\max_\Omega(\boldsymbol{\theta})$ and ii) $\partial\max_\Omega(\boldsymbol{\theta}) = \{\nabla\max_\Omega(\boldsymbol{\theta})\}$, since (3.5) has a unique solution. Therefore, Π_Ω is a **gradient mapping**. We illustrate $\max_\Omega$ and Π_Ω for various choices of Ω in Figure 3.4 (2-d), in Figure 3.5 on the 3-dimensional simplex, and in Figure 3.6 in a 3-d cross section.

Importance of strong convexity. Our β-strong convexity assumption on Ω plays a crucial role and should not be underestimated. Recall that a function $f\colon \mathbb{R}^d \to \mathbb{R}$ is β-strongly convex w.r.t. a norm $\|\cdot\|$ if and only if its conjugate f^* is $\frac{1}{\beta}$-smooth w.r.t. the dual norm $\|\cdot\|_*$ (Zălinescu, 2002, Corollary 3.5.11) (Kakade et al., 2012, Theorem 3). This is sufficient to ensure that $\max_\Omega$ is $\frac{1}{\gamma\beta}$-smooth, or, in other words, that it is differentiable everywhere and its gradient, Π_Ω, is $\frac{1}{\gamma\beta}$-Lipschitz continuous w.r.t. $\|\cdot\|_*$.

Training by backpropagation. In order to use Π_Ω in a neural network trained by backpropagation, two problems must be addressed for any regularizer Ω. The first is the **forward** computation: how to evaluate $\Pi_\Omega(\boldsymbol{\theta})$, i.e., how to solve the optimization problem in (3.5). The second is the **backward** computation: how to evaluate the Jacobian of $\Pi_\Omega(\boldsymbol{\theta})$, or, equivalently, the Hessian of $\max_\Omega(\boldsymbol{\theta})$. One of

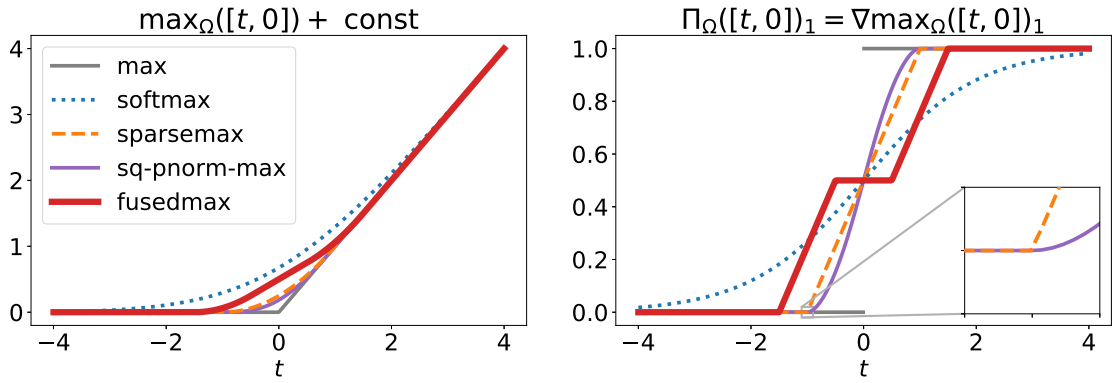


Figure 3.4: The proposed $\max_\Omega(\boldsymbol{\theta})$ operator up to a constant (left) and the $\Pi_\Omega(\boldsymbol{\theta})$ mapping (right), illustrated for $\boldsymbol{\theta} = [t, 0]$. In this case, $\max_\Omega(\boldsymbol{\theta})$ is hinge-shaped and $\Pi_\Omega(\boldsymbol{\theta})$ is sigmoid-shaped. Our framework recovers softmax and sparsemax.

our key contributions, presented in Section 3.4, is to show how to solve these two problems for general differentiable Ω, as well as for two structured regularizers: fused lasso and OSCAR.

3.3 Recovering Known Mappings and Characterizing Sparsity

Before deriving new attention mechanisms using our framework, we first show how our framework recovers softmax and sparsemax by careful choice of Ω.

Softmax. We choose $\Omega(\boldsymbol{p}) = \sum_{i=1}^d p_i \log p_i$, the negative Shannon entropy. The conjugate of the negative entropy restricted to the simplex is the log sum exp (Boyd and Vandenberghe, 2004, Example 3.25). Moreover, if $f(\boldsymbol{x}) = \gamma g(\boldsymbol{x})$ for $\gamma > 0$, then $f^*(\boldsymbol{y}) = \gamma g^*(\boldsymbol{y}/\gamma)$. We therefore get a closed-form expression: $\max_\Omega(\boldsymbol{\theta}) = \gamma \log \sum_{i=1}^d \exp(\theta_i/\gamma)$. Since the negative entropy is 1-strongly convex w.r.t. $\|\cdot\|_1$ over $\triangle^d$, we get that $\max_\Omega$ is $\frac{1}{\gamma}$-smooth w.r.t. $\|\cdot\|_\infty$. We obtain the well-known softmax,

with temperature parameter γ, by taking the gradient of $\max_\Omega(\boldsymbol{\theta})$,

$$\Pi_\Omega(\boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}/\gamma)}{\sum_{i=1}^{d} \exp(\theta_i/\gamma)}, \tag{softmax}$$

where $\exp(\boldsymbol{\theta}/\gamma)$ is evaluated element-wise. Note that some authors also call $\max_\Omega$ a "soft max." Although Π_Ω is really a soft *arg max*, we opt to follow the more popular terminology. When $\boldsymbol{\theta} = [t, 0]$, it can be checked that $\max_\Omega(\boldsymbol{\theta})$ reduces to the softplus (Dugas et al., 2001) and $\Pi_\Omega(\boldsymbol{x})_1$ to a sigmoid Figure 3.4.

Sparsemax. We choose $\Omega(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{p}\|_2^2$, also known as Moreau-Yosida regularization in proximal operator theory (Nesterov, 2005; Parikh and Boyd, 2014). Since $\frac{1}{2}\|\cdot\|_2^2$ is 1-strongly convex w.r.t. $\|\cdot\|_2$, we get that $\max_\Omega$ is $\frac{1}{\gamma}$-smooth w.r.t. $\|\cdot\|_2$. In addition, it is easy to verify that

$$\Pi_\Omega(\boldsymbol{\theta}) = \mathbf{P}_{\triangle^d}(\boldsymbol{\theta}/\gamma) = \operatorname*{arg\,min}_{\boldsymbol{p}\in\triangle^d} \|\boldsymbol{p} - \boldsymbol{\theta}/\gamma\|^2 . \tag{sparsemax}$$

This mapping was introduced in an ad-hoc manner by Martins and Astudillo (2016) and was named sparsemax, due to the fact that it is a sparse alternative to softmax. The Euclidean projection onto the simplex, $\mathbf{P}_{\triangle^d}$, enjoys exact $\mathcal{O}(d)$ algorithms (Held et al., 1974; Brucker, 1984; Condat, 2016). Following (Martins and Astudillo, 2016), the Jacobian of Π_Ω is

$$\boldsymbol{J}_{\Pi_\Omega}(\boldsymbol{\theta}) = \frac{1}{\gamma}\boldsymbol{J}_{\mathbf{P}_{\triangle^d}}(\boldsymbol{\theta}/\gamma) = \frac{1}{\gamma}\left(\operatorname{diag}(\boldsymbol{s}) - \boldsymbol{s}\boldsymbol{s}^\top/\|\boldsymbol{s}\|_1\right), \tag{3.7}$$

where $\boldsymbol{s} \in \{0, 1\}^d$ indicates the nonzero elements of $\Pi_\Omega(\boldsymbol{\theta})$. Since Π_Ω is Lipschitz continuous, by Rademacher's theorem Π_Ω is differentiable almost everywhere. For points where Π_Ω is not differentiable (*i. e.*, $\max_\Omega$ is not twice differentiable), we can take an arbitrary matrix in the set of Clarke's generalized Jacobians (Clarke, 1990), the convex hull of Jacobians of the form $\lim_{\boldsymbol{\theta}_t\to\boldsymbol{\theta}} \boldsymbol{J}_{\Pi_\Omega}(\boldsymbol{\theta}_t)$ (Martins and Astudillo, 2016).

A condition for sparsity. The closed form of softmax makes it obvious that the probabilities it produces can never be exactly zero, since exp is strictly positive. Moreover, quadratic problems over the simplex yield sparse solutions (Figure 2.2), justifying sparsemax. However, for an arbitrary convex Ω, it is not *a priori* obvious whether Π_Ω will be sparse or dense. The following proposition provides a necessary and sufficient condition for sparsity.

Proposition 3.1 *Let $\Omega : \triangle^d \to \mathbb{R}$ be a strictly convex function. The mapping Π_Ω covers the full simplex, i. e., $\Pi_\Omega(\mathbb{R}^d) = \triangle^d$, if and only if $\partial\Omega(\boldsymbol{p}) \neq \varnothing$ for any $\boldsymbol{p} \in \triangle^d$.*

Proof is given in Appendix A.1. Functions whose gradient "explodes" in the boundary of their domain (hence failing to meet the condition in Proposition 3.1) are called "essentially smooth" (Rockafellar, 1970); an example is the negative Shannon entropy. For such functions, Π_Ω maps only to the relative interior of $\triangle^d$, never attaining boundary points (Wainwright and Jordan, 2008). This prevents these functions from generating sparse attention mappings.

3.4 Algorithms for General Differentiable Regularizers

Before tackling more structured regularizers, we address in this section the case of general differentiable regularizer Ω. Because $\Pi_\Omega(\boldsymbol{\theta})$ involves maximizing (3.5), a concave function over the simplex, its solution can be found using off-the-shelf projected gradient solvers. Therefore, the main challenge is how to compute the Jacobian of Π_Ω. This is what we address in the next proposition.

Proposition 3.2 *Jacobian of Π_Ω for any twice differentiable Ω (backward pass)*

Assume that Ω is twice differentiable over $\triangle^d$ and that $\Pi_\Omega(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{p}\in\triangle^d} \boldsymbol{p}^\top\boldsymbol{\theta} - \gamma\Omega(\boldsymbol{p}) = \boldsymbol{p}^\star$ has been computed. Then the Jacobian of Π_Ω at $\boldsymbol{\theta}$, denoted $\boldsymbol{J}_{\Pi_\Omega}$, can be obtained by solving the system

$$(\boldsymbol{I} + \boldsymbol{A}(\boldsymbol{B} - \boldsymbol{I}))\, \boldsymbol{J}_{\Pi_\Omega} = \boldsymbol{A}, \tag{3.8}$$

where we defined the shorthands $\boldsymbol{A} \coloneqq \boldsymbol{J}_{\mathbf{P}_{\triangle^d}}(\boldsymbol{p}^\star - \gamma\nabla\Omega(\boldsymbol{p}^\star) + \boldsymbol{\theta})$ and $\boldsymbol{B} \coloneqq \gamma\boldsymbol{H}_\Omega(\boldsymbol{p}^\star)$.

The proof is given in Appendix A.2. Unlike recent work tackling argmin differentiation through matrix differential calculus on the Karush–Kuhn–Tucker (KKT) conditions (Amos and Kolter, 2017), our proof technique relies on differentiating the fixed point iteration $\boldsymbol{p}^* = \mathbf{P}_{\triangle^d}(\boldsymbol{p}^\star - \nabla\Omega(\boldsymbol{p}^\star) + \boldsymbol{\theta})$.

Efficient computation. When training networks by backpropagation, the Jacobian is only accessed via products with vectors, to obtain $\boldsymbol{d}_{\boldsymbol{\theta}} \coloneqq \boldsymbol{J}_{\Pi_\Omega}{}^\top \boldsymbol{d}_{\boldsymbol{p}}$. Therefore, we may directly solve $(\boldsymbol{I} + \boldsymbol{A}(\boldsymbol{B} - \boldsymbol{I}))\, \boldsymbol{d}_{\boldsymbol{\theta}} = \boldsymbol{A}\boldsymbol{d}_{\boldsymbol{p}}$. As a sparsemax Jacobian (Equation 3.7), $\boldsymbol{A}$ is row-and column-sparse, and uniquely defined by its sparsity pattern. By splitting the system into equations corresponding to zero and nonzero rows of $\boldsymbol{A}$, we obtain that the solution $\boldsymbol{d}_{\boldsymbol{\theta}}$ must have the same sparsity pattern as the row-sparsity of $\boldsymbol{A}$, therefore we only need to solve a subset of the system. From the fixed-point iteration $\boldsymbol{p}^\star = \mathbf{P}_{\triangle^d}(\boldsymbol{p}^\star - \nabla\Omega(\boldsymbol{p}^\star) + \boldsymbol{\theta})$, it follows that the row-sparsity of $\boldsymbol{A}$ is the same as the sparsity of the forward pass solution $\boldsymbol{p}^\star$. The Jacobian-vector product can thus be computed in $\mathcal{O}(\mathrm{nnz}(\boldsymbol{p}^\star)^3)$.

Example: squared p-norms. As a useful example of a differentiable function over the simplex, we consider squared p-norms: $\Omega(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{p}\|_p^2 = \left(\sum_{i=1}^d p_i^p\right)^{2/p}$, where $\boldsymbol{p} \in \triangle^d$ and $p \in (1, 2]$. For this choice of p, it is known that the squared

p-norm is strongly convex w.r.t. $\|\cdot\|_p$ (Ball et al., 1994). This implies that $\max_\Omega$ is $\frac{1}{\gamma(p-1)}$ smooth w.r.t. $\|\cdot\|_q$, where $\frac{1}{p} + \frac{1}{q} = 1$. We call the induced mapping function *sq-pnorm-max*:

$$\Pi_\Omega(\boldsymbol{\theta}) = \operatorname*{arg\,min}_{\boldsymbol{p} \in \triangle^d} \frac{\gamma}{2} \|\boldsymbol{p}\|_p^2 - \boldsymbol{p}^\top \boldsymbol{\theta}. \qquad \text{(sq-pnorm-max)}$$

The gradient and Hessian needed for Proposition 3.2 can be computed by $\nabla\Omega(\boldsymbol{p}) = \frac{\boldsymbol{p}^{p-1}}{\|\boldsymbol{p}\|_p^{p-2}}$ and

$$H_\Omega(\boldsymbol{p}) = \operatorname{diag}(\boldsymbol{d}) + \boldsymbol{u}\boldsymbol{u}^\top, \quad \text{where} \quad \boldsymbol{d} = \frac{(p-1)}{\|\boldsymbol{p}\|_p^{p-2}} \boldsymbol{p}^{p-2} \quad \text{and} \quad \boldsymbol{u} = \sqrt{\frac{(2-p)}{\|\boldsymbol{p}\|_p^{2p-2}}} \boldsymbol{p}^{p-1}, \quad (3.9)$$

with the exponentiation performed element-wise. For $p = 2$, sq-pnorm-max recovers sparsemax; it generally encourages sparse outputs. However, as can be seen in the zoomed box in Figure 3.4 (right), the transition between the extremes $\boldsymbol{p}^\star = [0, 1]$ and $\boldsymbol{p}^\star = [1, 0]$ can be smoother when $1 < p < 2$. Throughout our experiments, we use $p = 1.5$.

3.5 Fusedmax and Oscarmax: Clustered Attention

Fusedmax. For cases when the input is sequential and the order is meaningful, as is the case for many natural languages, we propose *fusedmax*, an attention mechanism based on the *fused lasso* (Tibshirani et al., 2005), also known as 1-d total variation (TV). Fusedmax encourages paying attention to **contiguous segments**, with equal weights within each one. It is expressed under our framework by choosing $\Omega(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{p}\|_2^2 + \lambda \sum_{i=1}^{d-1} |p_{i+1} - p_i|$, *i. e.*, the sum of a strongly convex term and of a 1-d TV penalty. It is easy to verify that this choice yields the mapping

$$\Pi_\Omega(\boldsymbol{\theta}) = \operatorname*{arg\,min}_{\boldsymbol{p} \in \triangle^d} \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{\theta}/\gamma\|^2 + \lambda \sum_{i=1}^{d-1} |p_{i+1} - p_i|. \qquad \text{(fusedmax)}$$

Oscarmax. For situations where the contiguity assumption may be too strict, we propose *oscarmax*, based on the OSCAR penalty (Bondell and Reich, 2008), to encourage attention weights to **merge into clusters with the same value**, regardless of position in the sequence. This is accomplished by replacing the 1-d TV penalty in fusedmax with an ∞-norm penalty on each pair of attention weights, *i. e.*, $\Omega(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{p}\|_2^2 + \lambda \sum_{i<j} \max(|p_i|, |p_j|)$. This results in the mapping

$$\Pi_\Omega(\boldsymbol{\theta}) = \operatorname*{arg\,min}_{\boldsymbol{p} \in \triangle^d} \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{\theta}/\gamma\|^2 + \lambda \sum_{i<j} \max(|p_i|, |p_j|). \tag{oscarmax}$$

Forward computation. Due to the $\boldsymbol{p} \in \triangle^d$ constraint, computing fusedmax/oscarmax does not seem trivial on first sight. The next proposition shows how to do so, without any iterative method.

Proposition 3.3 *Computing fusedmax and oscarmax (forward computation)*

$$\textit{fusedmax: } \Pi_\Omega(\boldsymbol{\theta}) = \mathbf{P}_{\triangle^d}\left(\mathbf{prox}_{TV}(\boldsymbol{\theta}/\gamma)\right), \textit{ where}$$
$$\mathbf{prox}_{TV}(\boldsymbol{\theta}) \coloneqq \operatorname*{arg\,min}_{\boldsymbol{p} \in \mathbb{R}^d} \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{\theta}\|^2 + \lambda \sum_{i=1}^{d-1} |p_{i+1} - p_i|.$$

$$\textit{oscarmax: } \Pi_\Omega(\boldsymbol{\theta}) = \mathbf{P}_{\triangle^d}\left(\mathbf{prox}_{OSC}(\boldsymbol{\theta}/\gamma)\right), \textit{ where}$$
$$\mathbf{prox}_{OSC}(\boldsymbol{\theta}) \coloneqq \operatorname*{arg\,min}_{\boldsymbol{p} \in \mathbb{R}^d} \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{\theta}\|^2 + \lambda \sum_{i<j} \max(|p_i|, |p_j|).$$

Here, $\mathbf{prox}_{\mathrm{TV}}$ and $\mathbf{prox}_{\mathrm{OSC}}$ indicate the proximal operators of 1-d TV and OSCAR, and can be computed **exactly**, as shown by Condat (2013) and Zeng and Figueiredo (2014), respectively. $\mathbf{P}_{\triangle^d}$ denotes the Euclidean projection onto the simplex, as in *sparsemax*, also computable exactly (Held et al., 1974; Brucker, 1984; Condat, 2016).

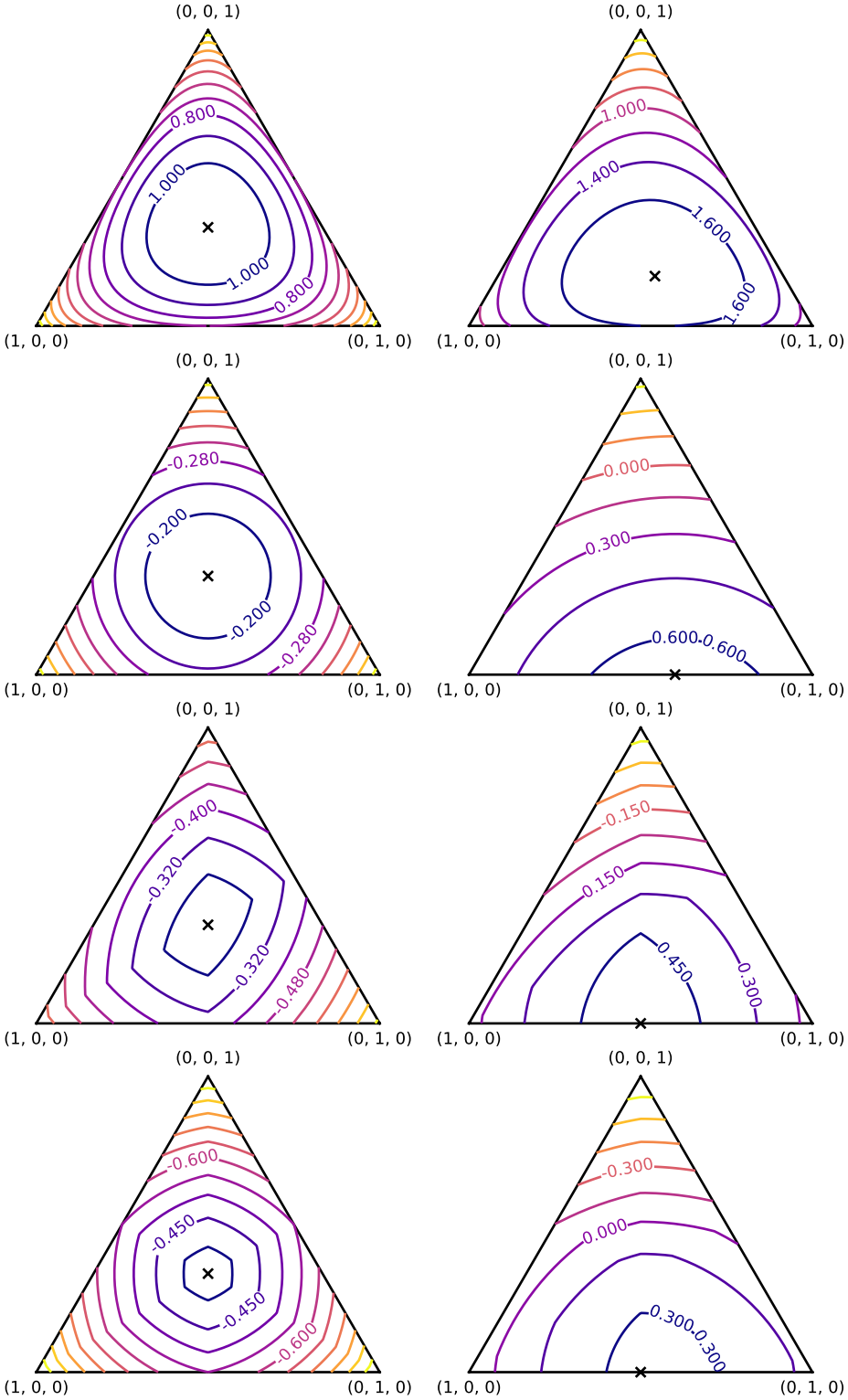


Figure 3.5: Contour plots of attention mechanisms on the simplex. From top to bottom: softmax, sparsemax, fusedmax and oscarmax. Left column: contours of $-\Omega$. Right column: contours of $f(\boldsymbol{p}) = \boldsymbol{p}^\top \boldsymbol{\theta} - \Omega(\boldsymbol{p})$, and the optimal $\boldsymbol{p}^\star$, for $\boldsymbol{\theta} = [.8, 1, 0]$.

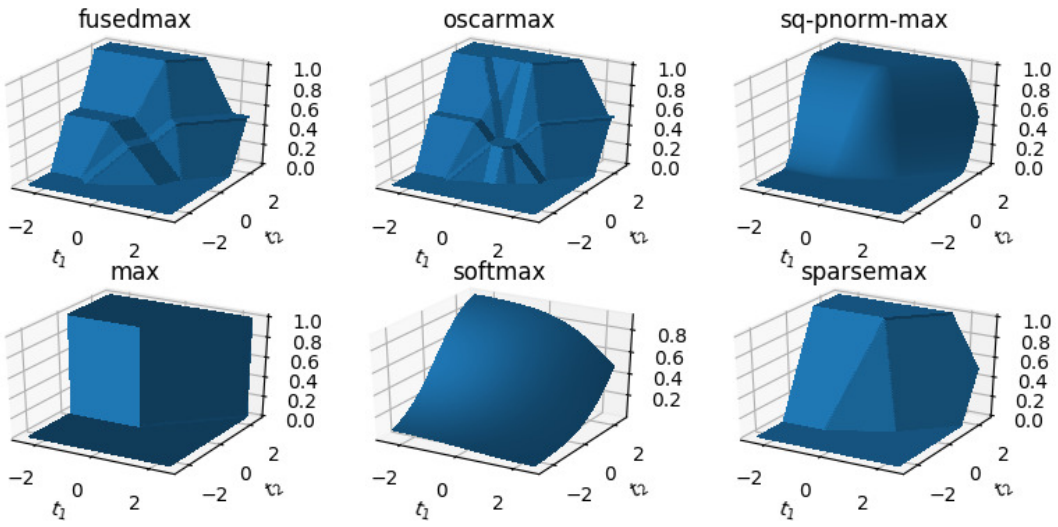


Figure 3.6: 3-d visualization of $\Pi_\Omega([t_1, t_2, 0])_2$ for several proposed and existing mappings Π_Ω. sq-pnorm-max with $p = 1.5$ resembles sparsemax but with smoother transitions. The proposed structured attention mechanisms, fusedmax and oscarmax, exhibit plateaus and ridges in areas where weights become fused together.

Proposition 3.3 shows that we can compute fusedmax and oscarmax using the composition of two functions, for which exact non-iterative algorithms exist. This is a surprising result, since the proximal operator of the sum of two functions is not, in general, the composition of the proximal operators of each function. The proof follows by showing that the indicator function of $\triangle^d$ satisfies the conditions of Yu (2013, Corollaries 4,5).

Groups induced by $\mathbf{prox}_{\mathrm{TV}}$ and $\mathbf{prox}_{\mathrm{OSC}}$. Let $\boldsymbol{z}^\star$ be the optimal solution of $\mathbf{prox}_{\mathrm{TV}}(\boldsymbol{\theta})$ or $\mathbf{prox}_{\mathrm{OSC}}(\boldsymbol{\theta})$. For $\mathbf{prox}_{\mathrm{TV}}$, we denote the group of **adjacent elements with the same value** as $z_i^\star$ by $G_i^\star$, $\forall i \in [d]$. Formally, $G_i^\star = [a, b] \cap \mathbb{N}$ with $a \leq i \leq b$ where a and b are the minimal and maximal indices such that $z_i^\star = z_j^\star$ for all $j \in G_i^\star$. For $\mathbf{prox}_{\mathrm{OSC}}$, we define $G_i^\star$ as the indices of **elements with the same absolute value** as $z_i^\star$, more formally $G_i^\star = \{j \in [d] \colon |z_i^\star| = |z_j^\star|\}$. Because $\mathbf{P}_{\triangle^d}(\boldsymbol{z}^\star) = \max(\boldsymbol{z}^\star - \tau, 0)$ for some $\tau \in \mathbb{R}$, fusedmax/oscarmax either shift a group's common

value or set all its elements to zero.

The parameter λ controls the trade-off between no fusion (sparsemax) and all elements fused into a single trivial group. While tuning λ may improve performance, we observe that $\lambda = 0.1$ (fusedmax) and $\lambda = 0.01$ (oscarmax) are sensible defaults that work well across all tasks; we report all our results using this setting.

Backward computation. We already know that the Jacobian of $\mathbf{P}_{\triangle^d}$ is the same as that of sparsemax with $\gamma = 1$. Then, by Proposition 3.3, if we know how to compute the Jacobians of $\mathbf{prox}_{\mathrm{TV}}$ and $\mathbf{prox}_{\mathrm{OSC}}$, we can obtain the Jacobians of fusedmax and oscarmax by straightforward application of the chain rule. However, although $\mathbf{prox}_{\mathrm{TV}}$ and $\mathbf{prox}_{\mathrm{OSC}}$ can be computed exactly, they lack analytical expressions. We next show that we can nonetheless compute their Jacobians efficiently, without needing to solve a system.

Proposition 3.4 *Jacobians of* $\mathbf{prox}_{TV}$ *and* $\mathbf{prox}_{OSC}$ *(backward computation)*

Assume $\boldsymbol{z}^\star = \mathbf{prox}_{TV}(\boldsymbol{\theta})$ *or* $\mathbf{prox}_{OSC}(\boldsymbol{\theta})$ *has been computed. Define the groups derived from* $\boldsymbol{z}^\star$ *as above. We have*

$$
\begin{aligned}
[\boldsymbol{J}\mathbf{prox}_{TV}(\boldsymbol{\theta})]_{i,j} &= \begin{cases} \frac{1}{|G_i^\star|} & \text{if } j \in G_i^\star, \\ 0 & \text{o.w.} \end{cases} \\
[\boldsymbol{J}\mathbf{prox}_{OSC}(\boldsymbol{\theta})]_{i,j} &= \begin{cases} \frac{\mathsf{sign}(z_i^\star z_j^\star)}{|G_i^\star|} & \text{if } j \in G_i^\star \text{ and } z_i^\star \neq 0, \\ 0 & \text{o.w.} \end{cases}
\end{aligned}
\tag{3.10}
$$

The proof is given in Appendix A.3. Clearly, the structure of these Jacobians permits efficient Jacobian-vector products. Note that $\mathbf{prox}_{\mathrm{TV}}$ and $\mathbf{prox}_{\mathrm{OSC}}$ are differentiable

everywhere except at points where groups change. For these points, the same remark as for sparsemax applies, and we can use Clarke's Jacobian.

3.6 Experimental Results

We evaluate our proposed projection operators on three NLP tasks:

- **Natural language inference:** a classification task on sentence pairs: a *premise* and a *hypothesis*. The model must predict whether a reader would assume the hypothesis follows from the premise, whether it contradicts it, or whether it is neutral to it (Dagan et al., 2009). Attention is used to select the relevant part of the *premise* when making the final prediction.
- **Sentence summarization (compression):** a monolingual sentence transduction task, aiming to transform a longer sentence into a shorter summary of it. We use a sequence-to-sequence encoder-decoder model, wherein attention is deployed at each step to select the relevant part of the input sentence.
- **Machine translation:** also a sentence transduction task, aiming to transform a sentence from one language into another.

Baseline. Our goal is to illustrate that the proposed sparse attention framework results in effective *drop-in* replacements for softmax across different tasks and architectures. As such, we use default parameter settings from existing implementations, intervening only by replacing *softmax* with our own implementation of *fusedmax*, *oscarmax*, and *pnormmax*. Compared to the well-established softmax baseline, we aim to demonstrate the sparsity, interpretability, and at least comparable accuracy when using sparse attention mechanisms.

attention	accuracy
softmax	81.66
sparsemax	82.39
fusedmax	**82.41**
oscarmax	81.76

Table 3.1: Natural language inference test accuracy on SNLI.

Natural language inference results. Textual entailment is the task of deciding, given a text T and an hypothesis H, whether a human reading T is likely to infer that H is true. We use the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015), a collection of 570,000 English sentence pairs. Each pair consists of a sentence and an hypothesis, manually labeled with one of the labels ENTAILMENT, CONTRADICTION, OR NEUTRAL.

We use a variant of the neural attention–based classifier proposed for this dataset by Rocktäschel et al. (2016). We strictly follow the same methodology as Martins and Astudillo (2016) in terms of implementation, hyperparameters, and grid search. We employ the CPU implementation provided by Martins and Astudillo (2016) and simply replace sparsemax with fusedmax/oscarmax; we observe that training time per epoch is essentially the same for each of the four attention mechanisms

Table 3.1 shows that, for this task, fusedmax reaches the highest accuracy, and oscarmax slightly outperforms softmax. Furthermore, fusedmax results in the most interpretable feature groupings: Figure 3.7 shows the weights of the neural network's attention to the text, when considering the hypothesis "No one is dancing." In this case, all four models correctly predicted that the text "A band is playing on stage at a concert and the attendants are dancing to the music," denoted along the x-axis, **contradicts** the hypothesis, although the attention weights differ. Notably, fusedmax identifies the meaningful segment "band is playing".

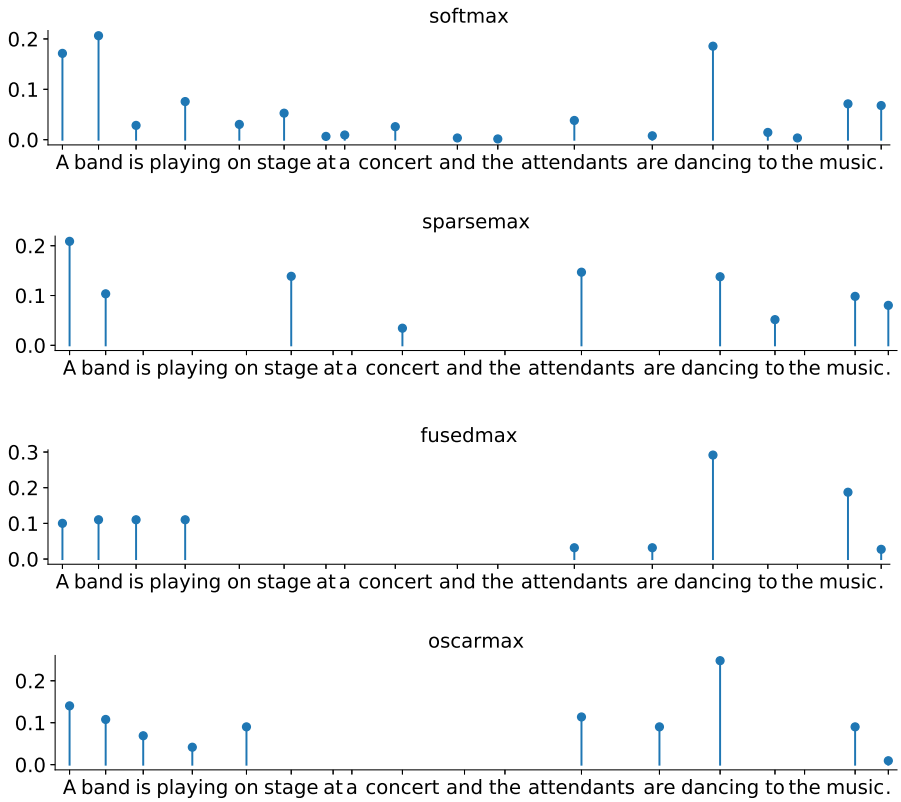


Figure 3.7: Attention weights for the contradicted hypothesis "No one is dancing."

Sentence summarization results. Attention mechanisms were recently explored for sentence summarization by Rush et al. (2015). To generate sentence-summary pairs at low cost, the authors proposed to use the title of a news article as a noisy summary of the article's leading sentence. They collected 4 million such pairs from the Gigaword dataset and showed that this seemingly simplistic approach leads to models that generalize surprisingly well.

We build on top of the OpenNMT-py package (Klein et al., 2017), an implementation of *seq2seq* neural models in PyTorch (PyTorch, 2017). We use the recommended default hyperparameters: unidirectional LSTM encoder/decoder, 500-dimensional word vectors and LSTM hidden representations, drop-out probability of 0.3, global attention, input-feeding (Luong et al., 2015), 13 training epochs with stochastic gradient updates (batches of size 64 and initial learning rate of 1, halved every

epoch after the 8^{th}). Weights (including word embeddings) are initialized uniformly over $[-0.1, 0.1]$, and gradients are normalized to have norm 5 if their norm exceeds this value. For test scores and visualizations, we use the model snapshot at the epoch with the highest validation set accuracy.

The only change required in the OpenNMT-py code is to replace *softmax* in the attention module with the proposed π_{Ω}. We follow the training methodology and data of Rush et al. (2015), reproducing comparable results to their softmax model.

Our evaluation follows (Rush et al., 2015): we use the standard DUC 2003 and 2004 dataset (500 news articles each paired with four different human-generated summaries) and a randomly held-out subset of Gigaword, released by Rush et al. (2015). We report results under the ROUGE-L metric. Our results, in Table 3.2, indicate that fusedmax is the best under nearly all metrics, always outperforming softmax; suggesting also that the sparse structured attention models are better at producing short summaries, benefiting less from maximum-length truncation, compared to softmax.

Machine translation results. Sequence-to-sequence neural machine translation (NMT) has recently become a strong contender in machine translation (Bahdanau et al., 2015; Luong et al., 2015). In NMT, attention weights can be seen as an *alignment* between source and translated words. To demonstrate the potential of our new attention mechanisms for NMT, we ran experiments on 10 language pairs, from the following sources:

- BENCHMARK: Training, validation, and test data from the NMT-Benchmark project (`http://scorer.nmt-benchmark.net/`). All languages have ~1M

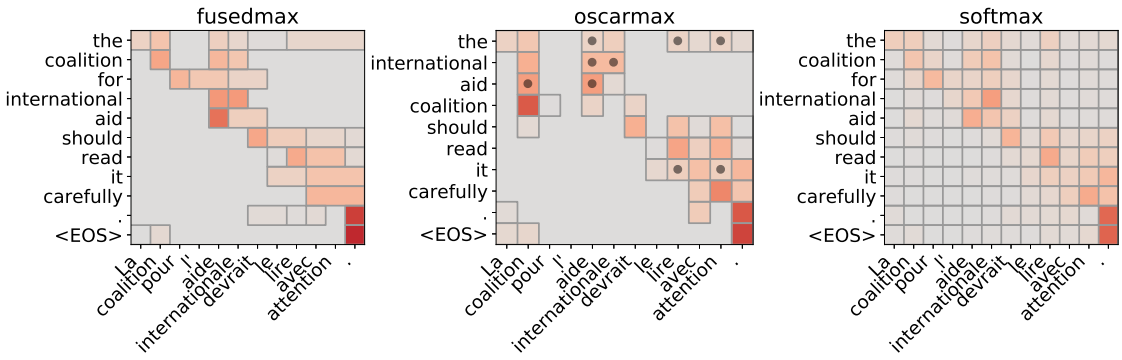


Figure 3.8: Attention weights for French to English translation. Within a row, weights grouped by oscarmax under the same cluster are denoted by "•". Here, oscarmax finds a slightly more natural English translation.

training sentence pairs, and equal validation and test sets of size 1K (French) and 2K (Italian, Dutch and Swedish).

- BENCHMARK$^+$: Training and validation data as above, but testing on all available *newstest* data. For Italian we use the 2009 data (~2.5K sentence pairs), and for French we concatenate 2009–2014 (~11K sentence pairs).
- WMT16, WMT17: Translation tasks at the first and second ACL Conferences for Machine Translation, available at `http://www.statmt.org/wmt16/translation-task.html` and `http://www.statmt.org/wmt17/translation-task.html`. Training, validation, and test sizes are, approximately, for Romanian 400K/2K/2K, for German 5.8M/6K/3K, for Finnish 2.6M/2K/2K, for Latvian 4.5M/2K/2K, and for Turkish 207K/1K/3K.

Once again, we use the *seq2seq* implementation from OpenNMT-py, with default hyperparameters. We use the preprocessing scripts from Moses (Koehn et al., 2007) for tokenization, and, where needed, SGML parsing. We limit source and target vocabulary sizes to 50K lower-cased tokens and prune sentences longer than 50 tokens at training time and 100 tokens at test time. We do not perform *recasing*.

We report BLEU scores in Table 3.3 and showcase the enhanced interpretability

induced by our proposed attention mechanisms in Figure 3.8. We find that all compared attention mechanisms are always within 1 BLEU score point of the best mechanism. This suggests that structured sparsity does not restrict accuracy. However, as illustrated in Figure 3.8, fusedmax and oscarmax often lead to more interpretable attention alignments, as well as to qualitatively different translations.

Summary of results. Generally, we observe that structured sparse attention performs is competitive as an out-of-the-box replacement to softmax. In particular, on sentence summarization, *fusedmax* appears to be a well-suited prior for the structure of the task. Across the board, structured sparsity leads to superior interpretability, as we illustrate with attention weight visualizations.

Computational considerations. OpenNMT-py with softmax attention is optimized for the GPU. Since sparsemax, fusedmax, and oscarmax rely on sorting operations, we implement their computations on the CPU for simplicity, keeping the rest of the pipeline on the GPU. However, we observe that, even with this context switching, the number of tokens processed per second was within $3/4$ of the softmax pipeline. For sq-pnorm-max, we observe that the projected gradient solver used in the forward pass, unlike the linear system solver used in the backward pass, could become a computational bottleneck. To mitigate this effect, we set the tolerance of the solver's stopping criterion to 10^{-2}.

	Truncated			Not truncated		
attention	P	R	F_1	P	R	F_1
DUC 2003						
softmax	29.57	20.67	23.87	30.40	20.80	24.23
sparsemax	29.59	20.58	23.89	30.37	20.68	24.21
fusedmax	**30.02**	**21.11**	**24.39**	**30.75**	**21.15**	**24.66**
oscarmax	29.64	20.78	24.02	30.40	20.87	24.32
sq-pnorm-max	29.45	20.50	23.78	30.23	20.56	24.07
DUC 2004						
softmax	30.54	21.00	24.47	30.59	21.13	24.55
sparsemax	30.99	21.57	24.96	31.03	21.64	25.02
fusedmax	**32.19**	**21.80**	**25.55**	**32.19**	**21.81**	**25.55**
oscarmax	31.89	21.46	25.14	31.91	21.51	25.17
sq-pnorm-max	31.42	21.55	25.08	31.46	21.63	25.13
Gigaword						
softmax	36.43	31.67	32.92	36.61	31.54	32.77
sparsemax	37.32	32.18	33.64	37.54	32.07	33.54
fusedmax	**37.44**	32.15	**33.69**	**37.68**	32.01	**33.59**
oscarmax	36.40	31.78	33.03	36.61	31.67	32.92
sq-pnorm-max	37.12	**32.37**	33.66	37.31	**32.26**	33.54

Table 3.2: Sentence summarization: ROUGE-L precision, recall and F-scores.

3.7 Discussion and Related Work

Smoothed max operators. Replacing the max operator by a differentiable approximation based on the log sum exp has been exploited in numerous works. Regularizing the max operator with a squared 2-norm is less frequent, but has been used to obtain a smoothed multiclass hinge loss (Shalev-Shwartz and Zhang, 2016) or smoothed linear programming relaxations for maximum a-posteriori inference (Meshi et al., 2015). Our work differs from these in mainly two aspects. First, we are less interested in the max operator itself than in its gradient, which we use as a mapping from $\mathbb{R}^d$ to Δ^d. Second, since we use this mapping in neural networks trained with backpropagation, we study and compute the mapping's Jacobian (the Hessian of a regularized max operator), in contrast with previous works.

	BENCHMARK				BENCHMARK+		WMT16	WMT17			
	fr	it	nl	sv	fr	it	ro	de	fi	lv	tr
from English											
softmax	36.94	37.20	36.12	34.97	27.13	**24.86**	17.71	22.32	14.54	11.02	**11.95**
sparsemax	37.03	37.21	36.12	**35.09**	26.99	24.49	17.61	**22.43**	**14.85**	11.07	11.66
fusedmax	37.08	36.73	36.04	34.30	26.89	24.47	17.19	22.25	14.28	**11.27**	11.32
oscarmax	36.66	36.89	35.96	34.86	27.02	24.76	17.26	22.42	14.02	11.19	11.63
sq-pnorm-max	**37.16**	**37.39**	**36.21**	34.63	**27.25**	24.56	**17.80**	—	14.45	—	11.58
to English											
softmax	36.79	39.95	40.06	37.96	25.72	25.37	17.86	**25.82**	15.11	13.60	11.78
sparsemax	**36.91**	40.13	40.25	38.09	25.97	25.62	17.46	25.76	14.95	13.59	**12.04**
fusedmax	36.64	39.64	39.87	37.83	25.72	25.41	**18.29**	25.58	15.08	13.53	11.91
oscarmax	36.90	40.05	40.17	**38.12**	**26.13**	25.65	17.89	25.69	14.94	**13.71**	11.70
sq-pnorm-max	36.84	**40.23**	**40.48**	**38.12**	25.72	**25.70**	17.44	—	**15.20**	—	11.93

Table 3.3: Neural machine translation results: tokenized BLEU test scores.

Interpretability, structure and sparsity in neural networks. Providing interpretations alongside predictions is important for accountability, error analysis and exploratory analysis, among other reasons. Toward this goal, several recent works have been relying on visualizing hidden layer activations (İrsoy, 2017; Li et al., 2016) and the potential for interpretability provided by attention mechanisms has been noted in multiple works (Bahdanau et al., 2015; Rocktäschel et al., 2016; Rush et al., 2015). Our work aims to fulfill this potential by providing a unified framework upon which new interpretable attention mechanisms can be designed, using well-studied tools from the field of structured sparse regularization.

Explanation generation. Selecting contiguous text segments for model interpretations is explored by Lei et al. (2016), where an *explanation generator* network is proposed for justifying predictions. This generator also uses the fused lasso, but it is not an attention mechanism and has its own parameters to be learned. Furthermore, their approach sidesteps the need to backpropagate through the fused lasso, unlike our work, by using a stochastic training approach. In contrast, our attention mechanisms are deterministic and **drop-in replacements** for softmax. In consequence, our mechanisms can be coupled with research building on top of softmax, for example incorporating soft prior knowledge about NMT alignment into attention through penalties on the attention weights (Cohn et al., 2016).

Structured attention networks. A different way to incorporate structure into attention is to use the posterior marginal probabilities from a conditional random field as attention weights (Kim et al., 2017). While this approach takes into account structural correlations, the marginal probabilities are generally dense and different from each other. Our proposed mechanisms produce sparse and clustered atten-

tion weights, a visible benefit in interpretability. We revisit structured attention networks in Chapter 4, where we develop a sparse inference technique that results in more interpretable and more widely applicable structured attention networks.

Sparsity in neural networks. Sparsity-inducing penalties have been used to obtain convex relaxations of neural networks (Bengio et al., 2005) or to compress models (Liu et al., 2015; Wen et al., 2016; Scardapane et al., 2017). These works differ from ours, in that sparsity is enforced on the network **parameters**, while we produce sparse and structured **outputs** from neural attention layers.

Submodular optimization. Obozinski and Bach (2016) proposed a framework for structured sparsity–inducing penalties using functions of sets. This strategy allows practitioners to characterize the properties of the desired support of the solutions, and, in specific cases, uncovers lasso, group lasso, OWL and OSCAR penalties. The authors provide generic algorithms for typical optimization problems involving such penalties, which correspond to the *forward pass* in our framework. Further study into the necessary *backward pass* of proximity operators or min-norm solutions would result in a very appealing strategy for flexible, user-defined latent structured sparsity. Recently, Djolonga and Krause (2017) provided significant results in this direction, characterizing the Jacobians of min-norm optimization of submodular functions, which constitute a subset of the family supported by Obozinski and Bach (2016).

3.8 Chapter Summary

We proposed in this chapter a unified regularized framework upon which new attention mechanisms can be designed. To enable such mechanisms to be used in a neural network trained by backpropagation, we demonstrated how to carry out forward and backward computations for general differentiable regularizers. We further developed two new structured attention mechanisms, *fusedmax* and *oscarmax*, and demonstrated that they enhance interpretability while achieving comparable or better accuracy on three diverse and challenging tasks: textual entailment, machine translation, and summarization.

The usefulness of a differentiable mapping from real values to the simplex or to $[0, 1]$ with sparse or structured outputs goes beyond attention mechanisms. We expect that our framework will be useful to sample from categorical distributions using the Gumbel trick (Jang et al., 2017; Maddison et al., 2017), as well as for conditional computation (Bengio et al., 2013) or differentiable neural computers (Graves et al., 2014, 2016).

CHAPTER 4

SPARSEMAP: DIFFERENTIABLE SPARSE STRUCTURED INFERENCE

In the previous chapter, we developed a general strategy for empowering neural networks with *structured sparsity* in their hidden representations. In contrast, we next switch gears toward thinking about *structure* in the form of *combinatorial objects*, *i. e.*, objects formed from discrete mathematical objects that combine together in numerous but constrained ways. Across application domains, but especially in NLP, many objects of interest can be represented by combinatorial structures: syntactic and dependency trees, sequential labelings, relations between entities in a text. These objects could be desired outputs of machine learning tasks, as well as intermediate representations in deep pipelines.

In this chapter, we introduce SparseMAP, a new method for **sparse structured inference**, whose solutions are sparse combinations of a small number of global structures. Thanks to this sparsity, gradient backpropagation is efficient regardless of the structure, enabling us to augment deep neural networks with generic and sparse **structured hidden layers**.

This chapter is based on part of (Niculae et al., 2018).

4.1 Structured Inference Preliminaries

To set the groundwork, we review some fundamental concepts on **structured inference**: the problem of finding a posterior distribution over discrete, combinatorial structures such as trees, sequences, or alignments (Bakır et al., 2007; Smith, 2011; Nowozin et al., 2014).

A typical structured prediction problem, using the notations of Section 2.1, involves predicting, given an input x, a structured object $y \in \mathcal{Y}$. The "catch" is that the set of possible outcomes $\mathcal{Y}$ is very large, typically exponential in the size of x. This rules out naïve iteration over the possible outputs. For instance, a relatively short, five-word sentence has 1296 possible dependency trees, while a ten-word sentence has more than two billion. In consequence, unlike typical classification problems, in structured prediction we cannot compute $\sigma(x, y)$ for every $y \in \mathcal{Y}$. This means even the innocent-looking prediction function

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \theta(y; x)$$

becomes difficult to evaluate, due to the size of $\mathcal{Y}$. For this reason, structured inference is often sidestepped by greedy search, factorization assumptions, or continuous relaxations (Belanger and McCallum, 2016).

In this section, we present some of the terminology and factorization assumptions used in structured inference. For simplicity, we abstract away the input x and focus on the scores $\theta(y)$. Such functions defined on the discrete set of all structures $\mathcal{Y}$ can also be seen as vectors $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{Y}|}$. We use either notation when convenient, *e. g.*, we may index into $|\mathcal{Y}|$-dimensional vectors as $\theta(y)$ instead of θ_y, for emphasis.

Probabilistic models and exponential families. The scores $\boldsymbol{\theta}$ can be used to define a probability distribution over $\mathcal{Y}$, reflecting the degree to which we believe object y should be preferred to the others

$$p(y) = \frac{\exp \theta(y)}{\sum_{y' \in \mathcal{Y}} \exp \theta(y')}. \tag{4.1}$$

Similarly, we sometimes think of such a probability distribution as a vector $\boldsymbol{p} \in \triangle^{|\mathcal{Y}|}$. The denominator $Z = \sum_{y' \in \mathcal{Y}} \exp \theta(y')$ is called the **partition function** and it

reveals the challenge in this probabilistic approach, as it is a typically intractable exponential sum. The family of such probability distributions, parametrized by the score vector θ, is an **exponential family**.

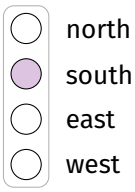
Factorization assumptions and graphical models. To circumvent the exponential size of θ, we commonly assume that the score of an object y decomposes, or *factors*, over some smaller *parts* of y, $\mathcal{P}(y)$. This reflects the *combinatorial* assumption that objects are combinations of smaller, discrete units. $\mathcal{P}(y)$ is further divided, typically, into *variable* assignments (unaries) $\mathcal{U}(y)$—encoding the components of the desired output, and possibly *higher-order factors* $\mathcal{V}(y)$—encoding correlations between multiple variables. We thus have

$$\begin{aligned} \theta(y) &\coloneqq \sum_{p \in \mathcal{P}(y)} \eta_p = \boldsymbol{a}_y^\top \boldsymbol{\eta} \\ &\coloneqq \sum_{i \in \mathcal{U}(y)} \eta_{u,i} + \sum_{j \in \mathcal{V}(y)} \eta_{v,j} = \boldsymbol{m}_y^\top \boldsymbol{\eta}_u + \boldsymbol{n}_y^\top \boldsymbol{\eta}_v \end{aligned} \tag{4.2}$$

where we introduced the matrix $\boldsymbol{A} \in \mathbb{R}^{d \times D}$ defined as $\boldsymbol{A}^\top \coloneqq [\boldsymbol{M}^\top, \boldsymbol{N}^\top]$ with $d = d_u + d_v \ll D, d_u = |\mathcal{U}(y)|, d_v = |\mathcal{V}(y)|$, and where $\boldsymbol{\eta} = [\boldsymbol{\eta}_u, \boldsymbol{\eta}_v]$ is a tractable, low-dimensional parametrization. Given a structure y, the column $\boldsymbol{m}_y$ is a vector encoding the variable assignments which characterize y, and the corresponding $\boldsymbol{n}_y$ indicates which higher-order factors are present in y. This formalism not only makes the parametrization tractable, but also allows sharing parts of scores between structures that are similar, *i. e.*, that have parts in common. For instance, it seems reasonable to assume that two dependency trees differing only in one arc could share the computation of most of their scores. Figure 4.1 illustrates this construction on several commonly-used types of structure, described more rigorously at the end of this section.

structure	M	N

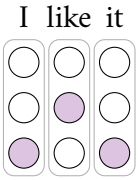
One-of-K (XOR)



	y_n	y_s	y_e	y_w
north	1	0	0	0
south	0	1	0	0
east	0	0	1	0
west	0	0	0	1

[]

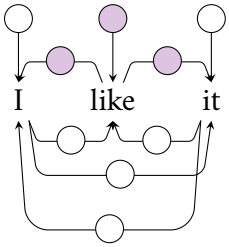
Sequence tagging



	y_{NNN}	y_{NNV}		y_{PVP}	
I=N	1	1		0	
I=V	0	0		0	
I=P	0	0		1	
like=N	1	1		0	
like=V	0	0	...	1	...
like=P	0	0		0	
it=N	1	0		0	
it=V	0	1		0	
it=P	0	0		1	

	y_{NNN}	y_{NNV}		y_{PVP}	
I,like=NN	1	1		0	
I,like=NV	0	0		0	
I,like=NP	0	0		0	
I,like=VN	0	0		0	
I,like=VV	0	0	...	0	...
I,like=VP	0	0		0	
I,like=PN	0	0		0	
I,like=PV	0	0		1	
I,like=PP	0	0		0	
like,it=NN	1	0		0	
		...			
like,it=VP	0	0		1	
		...			

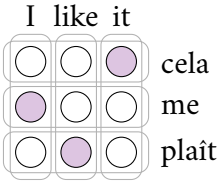
Non-projective dependency parsing



⋆→I	1		0	0
like→I	0		1	1
it→I	0		0	0
⋆→like	0		1	1
I→like	1	...	0	0 ...
it→like	0		0	0
⋆→it	0		0	0
I→it	0		1	0
like→it	1		0	1

[]

Linear assignment



	y_{123}	y_{132}	y_{213}	y_{231}	y_{312}	y_{321}
I–cela	1	1	0	0	0	0
I–me	0	0	1	1	0	0
I - plaît	0	0	0	0	1	1
like–cela	0	0	1	0	1	0
like–me	1	0	0	0	0	1
like - plaît	0	1	0	1	0	0
it–cela	0	0	0	1	0	1
it–me	0	1	0	0	1	0
it - plaît	1	0	1	0	0	0

[]

Figure 4.1: Illustration of useful structures, along with their matrix representation.

The marginal polytope. Each probability distribution generated by Equation 4.1 can be seen as a discrete vector $\boldsymbol{p} \in \triangle^D$. The lower-dimensional parametrization through $\boldsymbol{A}$ in Equation 4.2 provides a way to replace the D-dimensional simplex with an only d-dimensional polytope with D vertices, one for each possible structure

$$\mathcal{M}_{\boldsymbol{A}} = \{\boldsymbol{\mu} : \boldsymbol{\mu} = \boldsymbol{A}\boldsymbol{p}; \boldsymbol{p} \in \triangle^D\}. \tag{4.3}$$

This set is called the marginal polytope of the structured model. A point $\boldsymbol{\mu} \in \mathcal{M}_{\boldsymbol{A}}$ can be seen as the expected structure under a particular distribution p

$$\boldsymbol{\mu} = \mathbb{E}_{Y \sim p}[\boldsymbol{a}_Y] = \sum_{y \in \mathcal{Y}} \boldsymbol{a}_y p(y) = \boldsymbol{A}\boldsymbol{p}.$$

An important result from Wainwright and Jordan (2008) shows that any point $\boldsymbol{\mu}$ in the relative interior of $\mathcal{M}_{\boldsymbol{A}}$ (*i. e.*, excluding the boundary), there exists some $\boldsymbol{\eta}$ such that the exponential distribution with parameters $\boldsymbol{\theta} = \boldsymbol{A}^\top \boldsymbol{\eta}$ has mean representation $\mathbb{E}_Y[\boldsymbol{a}_Y] = \boldsymbol{\mu}$.

Inference. The two fundamental questions that we might ask in a structured model are:

- What is the most likely structure?
- What is a representation of the average structure?

Identifying the most likely structure is known as **maximum *a posteriori* (MAP) inference**. This can be seen as a D-dimensional max, as in Chapter 3

$$\max_{y \in \mathcal{Y}} \theta(y) = \max_{\boldsymbol{p} \in \triangle^D} \boldsymbol{\theta}^\top \boldsymbol{p}. \tag{4.4}$$

We may rewrite this as an k-dimensional optimization over $\mathcal{M}_A$[1]

$$\begin{aligned} \mathsf{MAP}_A(\boldsymbol{\eta}) &\coloneqq \operatorname*{arg\,max}_{\substack{\boldsymbol{u}:=\boldsymbol{M}\boldsymbol{p} \\ \boldsymbol{p}\in\triangle^D}} \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} \\ &= \operatorname*{arg\,max}_{\boldsymbol{u}:\,[\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}_A} {\boldsymbol{\eta}_u}^\top \boldsymbol{u} + {\boldsymbol{\eta}_v}^\top \boldsymbol{v} \end{aligned} \tag{4.5}$$

Since it is essentially a D-dimensional $\mathsf{arg\,max}$, MAP is piecewise constant and discontinuous: small changes to the input $\boldsymbol{\eta}$ typically do not change the solution, except at breaking points, where the change happens abruptly.

Finding a representation of the average structure is known as **marginal inference**, and is equivalent to applying negative entropy regularization:

$$\begin{aligned} \mathsf{Marginal}_A(\boldsymbol{\eta}) &\coloneqq \mathbb{E}_Y[\boldsymbol{m}_Y] \\ &= \operatorname*{arg\,max}_{\substack{\boldsymbol{u}:=\boldsymbol{M}\boldsymbol{p} \\ \boldsymbol{p}\in\triangle^D}} \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} + H(\boldsymbol{p}) \\ &= \operatorname*{arg\,max}_{\boldsymbol{u}:\,[\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}_A} {\boldsymbol{\eta}_u}^\top \boldsymbol{u} + {\boldsymbol{\eta}_v}^\top \boldsymbol{v} + H_A(\boldsymbol{u},\boldsymbol{v}). \end{aligned} \tag{4.6}$$

Marginal inference is differentiable, but may be more difficult to compute; the entropy $H_A(\boldsymbol{u},\boldsymbol{v}) = H(\boldsymbol{p})$ itself lacks a closed form (Wainwright and Jordan, 2008, Section 4.1.2). Gradient backpropagation is available only to specialized problem instances, *e.g.*, those solvable by dynamic programming (Li and Eisner, 2009; Mensch and Blondel, 2018). The entropic term regularizes $\boldsymbol{y}$ toward more uniform distributions, resulting in strictly dense solutions, just like in the case of $\mathsf{softmax}$.

Examples of structures. We now provide a description of important structures, which will show up later in experiments. Refer to Figure 4.1 for visualization cues and concrete examples of the $\boldsymbol{M}$ and $\boldsymbol{N}$ matrices.

[1]We use the notation $\mathsf{arg\,max}_{\boldsymbol{u}:\,[\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}}$ to convey that the maximization is over both $\boldsymbol{u}$ and $\boldsymbol{v}$, but only $\boldsymbol{u}$ is returned. Separating the variables as $[\boldsymbol{u},\boldsymbol{v}]$ loses no generality and allows us to isolate the unary posteriors $\boldsymbol{u}$ as the return value of interest.

- **One-of-K (XOR) factor.** Consider k binary variables, which could generally have 2^k joint configurations. An XOR factor enforces the hard constraint that exactly one of the variables can be on, leaving exactly k valid configurations. The XOR factor is often used in combination with other factors to enforce uniqueness, *e. g.*, in frame-semantic parsing, where each semantic role must be filled by exactly one span (Das et al., 2014). But the XOR factor is interesting in isolation: up to a permutation of the variables, we have $\boldsymbol{A} = \boldsymbol{I}$. Therefore, MAP reduces to arg max, and marginal inference is equivalent to softmax. This provides a bridge between the unstructured case, as studied in Chapter 3, and the structured case we examine in this chapter.

- **Sequence tagging.** Consider a sequence of n items, each assigned one out of a possible m tags. In this case, a global structure y is a joint assignment of tags $(t_1, \cdots, t_n)$. The matrix $\boldsymbol{M}$ is nm-by-m^n–dimensional, with columns $\boldsymbol{m}_y \in \{0,1\}^{nm} := [\boldsymbol{e}_{t_1}, ..., \boldsymbol{e}_{t_n}]$ indicating which tag is assigned to each variable in the global structure y. $\boldsymbol{N}$ is nm^2-by-m^n–dimensional, with $\boldsymbol{n}_y$ encoding the transitions between consecutive tags, *i. e.*, $\boldsymbol{n}_y(i, a, b) := \imath[t_{i-1} = a \;\&\; t_i = b]$. The Viterbi algorithm provides MAP inference and forward-backward provides marginal inference (Rabiner, 1989).

- **Non-projective dependency parsing.** Consider a sentence of length n. Here, a structure y is a dependency tree: a rooted spanning tree over the n^2 possible arcs (for example, the arcs above the sentences in Figure 6.2). Each column $\boldsymbol{m}_y \in \{0,1\}^{n^2}$ encodes a tree by assigning a 1 to its arcs. $\boldsymbol{N}$ is empty, $\mathcal{M}_{\boldsymbol{A}}$ is known as the *arborescence polytope* (Martins et al., 2009). MAP inference may be performed by *maximal arborescence* algorithms (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005a), and the Matrix-Tree theorem (Kirchhoff, 1847) provides a way to perform marginal inference

(Koo et al., 2007; Smith and Smith, 2007; McDonald and Satta, 2007).

- **Linear assignment.** Consider a one-to-one matching (linear assignment) between two sets of n nodes. A global structure y is a n-permutation, and a column $\boldsymbol{m}_y \in \{0,1\}^{n^2}$ can be seen as a flattening of the corresponding permutation matrix. Again, $\boldsymbol{N}$ is empty. $\mathcal{M}_A$ is the Birkhoff polytope (Birkhoff, 1946), and MAP inference can be performed by, *e.g.*, the Hungarian algorithm (Kuhn, 1955) or the Jonker-Volgenant algorithm (Jonker and Volgenant, 1987). Crucially, marginal inference is known to be #P-complete (Valiant, 1979; Taskar, 2004, Section 3.5). (In fact, it is equivalent to the problem used to initially define the #P-complete complexity class.) As such, using matchings as latent variables remains an open problem.

4.2 Sparse Structured Inference

We propose a new inference strategy, dubbed SparseMAP, which encourages **sparsity** in the structured representations. Namely, we seek solutions explicitly expressed as a combination of a small, enumerable set of global structures.

SparseMAP is a twofold generalization: first, as a structured extension of the sparsemax transformation (Martins and Astudillo, 2016); second, as a continuous yet sparse relaxation of MAP inference. Our framework departs from the two most common inference strategies in structured prediction: MAP, which returns the highest-scoring structure, and marginal inference, which yields a dense probability distribution over structures. Neither of these strategies is fully satisfactory: for latent structure models, marginal inference is appealing, since it can represent uncertainty and, unlike MAP inference, it is continuous and differentiable, hence

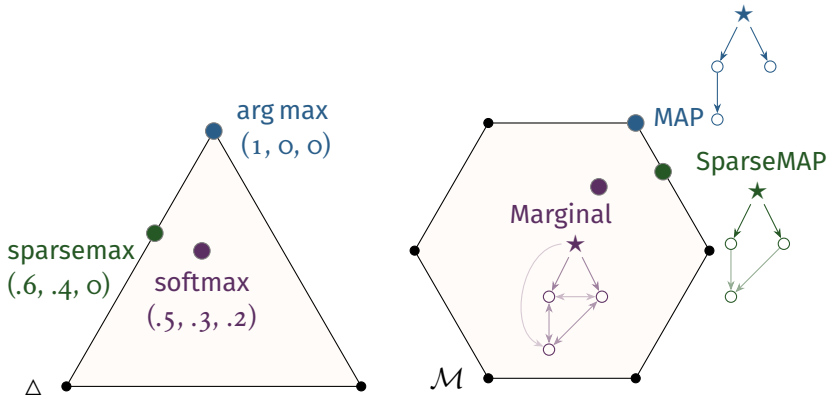


Figure 4.2: Geometrical interpretation of various inference strategies. Left: in the unstructured case, softmax and sparsemax can be interpreted as regularized, differentiable arg max approximations; softmax returns dense solutions while sparsemax favors sparse ones. Right: in this work, we extend this view to *structured inference*, which consists of optimizing over a polytope $\mathcal{M}$, the convex hull of all possible structures (depicted: the arborescence polytope, whose vertices are trees). We introduce SparseMAP as a structured extension of sparsemax: it is situated in between MAP inference, which yields a single structure, and marginal inference, which returns a dense combination of structures.

amenable for use in structured hidden layers in neural networks (Kim et al., 2017). It has, however, several limitations. For one, there are useful problems for which MAP is tractable, but marginal inference is not, *e. g.*, linear assignment (Valiant, 1979; Taskar, 2004). Even when marginal inference is available, case-by-case derivation of the backward pass is needed, sometimes producing fairly complicated algorithms, *e. g.*, second-order expectation semirings (Li and Eisner, 2009). Finally, while capable of representing uncertainty, marginal inference is *dense*, *i. e.*, it assigns non-zero probabilities to all structures and cannot *completely rule out* irrelevant ones. This can be statistically and computationally wasteful, as well as qualitatively harder to interpret.

SparseMAP. Armed with the parallel between structured inference and regularized max operators uncovered in Section 4.1, we are now ready to introduce SparseMAP,

a novel inference optimization problem which returns sparse solutions.

We introduce SparseMAP by regularizing the MAP inference problem in Equation 4.5 with a squared ℓ_2 penalty on the returned posteriors, *i. e.*, $\frac{1}{2}\|\boldsymbol{u}\|_2^2$. Denoting, as above, $\boldsymbol{\theta} \coloneqq \boldsymbol{A}^\top \boldsymbol{\eta}$, SparseMAP is the quadratic optimization problem

$$\begin{aligned} \mathsf{SparseMAP}_{\boldsymbol{A}}(\boldsymbol{\eta}) \coloneqq & \operatorname*{arg\,max}_{\substack{\boldsymbol{u}:=\boldsymbol{M}\boldsymbol{p} \\ \boldsymbol{p}\in\triangle^D}} \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} - \frac{1}{2}\|\boldsymbol{M}\boldsymbol{p}\|_2^2 \\ & = \operatorname*{arg\,max}_{\boldsymbol{u}:\ [\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}_{\boldsymbol{A}}} {\boldsymbol{\eta}_u}^\top \boldsymbol{u} + {\boldsymbol{\eta}_v}^\top \boldsymbol{v} - \frac{1}{2}\|\boldsymbol{u}\|_2^2 . \end{aligned} \tag{4.7}$$

The quadratic penalty replaces the entropic penalty from marginal inference (Equation 4.6), which pushes the solutions to the strict interior of the marginal polytope. In consequence, SparseMAP favors sparse solutions from the boundary of the marginal polytope $\mathcal{M}_{\boldsymbol{A}}$, as illustrated in Figure 4.2. For the structured prediction problems mentioned in Section 4.1, SparseMAP is able to return, for example, a sparse combination of sequence labelings, parse trees, or matchings. For the case of the XOR factor, when $\boldsymbol{A} = \boldsymbol{I}$, it can easily be verified that SparseMAP reduces to sparsemax. Moreover, the strongly convex regularization on $\boldsymbol{u}$ ensures that SparseMAP has a unique solution and is differentiable almost everywhere.

4.3 Computing SparseMAP

We now tackle the optimization problem in Equation 4.7, *i. e.*, computing SparseMAP. Although the optimization problem in question is a quadratic program over a polytope, even describing it in standard form is infeasible, since enumerating the exponentially-large set of vertices is infeasible. This prevents direct application of generic QP solvers, in particular the differentiable QP solver of Amos and

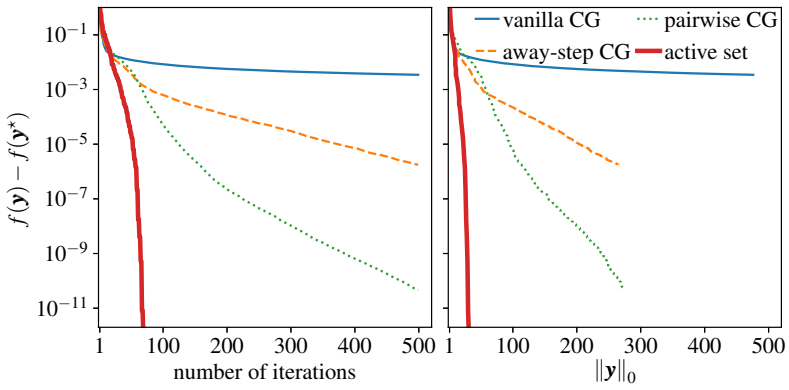


Figure 4.3: Comparison of solvers on the SparseMAP optimization problem for a tree factor with 20 nodes. The active set solver converges much faster and to a much sparser solution.

Kolter (2017). We instead focus on SparseMAP solvers that involve a sequence of MAP problems as a subroutine—this makes SparseMAP widely applicable, given the availability of MAP implementations for various structures. We discuss two such methods, one based on the conditional gradient algorithm and another based on the active set method for quadratic programming.

Conditional gradient. One family of such solvers is based on the *conditional gradient* (CG) algorithm (Frank and Wolfe, 1956; Lacoste-Julien and Jaggi, 2015), considered in prior work for solving approximations of the marginal inference problem (Belanger et al., 2013; Krishnan et al., 2015).

Each step must solve a linearized subproblem. Denote by f the SparseMAP objective from Equation 4.7,

$$f(\boldsymbol{u}, \boldsymbol{v}) \coloneqq {\boldsymbol{\eta}_u}^\top \boldsymbol{u} + {\boldsymbol{\eta}_v}^\top \boldsymbol{v} - \frac{1}{2} \|\boldsymbol{u}\|_2^2 .$$

The gradients of f with respect to the two variables are

$$\nabla_{\boldsymbol{u}} f(\boldsymbol{u}', \boldsymbol{v}') = \boldsymbol{\eta}_u - \boldsymbol{u}', \qquad \nabla_{\boldsymbol{v}} f(\boldsymbol{u}', \boldsymbol{v}') = \boldsymbol{\eta}_v.$$

A linear approximation to f around a point $[\boldsymbol{u}', \boldsymbol{v}']$ is given by the first-order Taylor expansion

$$\hat{f}(\boldsymbol{u}, \boldsymbol{v}) \coloneqq (\nabla_{\boldsymbol{u}} f)^\top \boldsymbol{u} + (\nabla_{\boldsymbol{v}} f)^\top \boldsymbol{v} = (\boldsymbol{\eta}_u - \boldsymbol{u}')^\top \boldsymbol{u} + \boldsymbol{\eta}_v{}^\top \boldsymbol{v},$$

Minimizing $\hat{f}$ over $\mathcal{M}$ is exactly the same as `MAP` inference with adjusted variable scores $\boldsymbol{\eta}_u - \boldsymbol{u}'$, where $\boldsymbol{u}'$ is the current estimate of the solution. Intuitively, at each step we seek a high-scoring structure while penalizing sharing variables with structures that have already been selected. Vanilla CG simply adds the new structure to the active set at every iteration, with a coefficient selected by line search. Pairwise and away-step CG are variants that trade off between moving in the direction of the new structure and moving away from one of the already-selected structures. A full specification of the CG variants employed is provided in Appendix B.1.

Active set method. Importantly, the `SparseMAP` problem in Equation 4.7 has quadratic curvature, which the general CG algorithms may not optimally leverage. For this reason, we consider the active set method for constrained QPs (Nocedal and Wright, 1999, Chapters 16.4 & 16.5), a generalization of Wolfe's min-norm point algorithm (Wolfe, 1976), also used in structured prediction for the quadratic subproblems by Martins et al. (2015). The procedure, described in Algorithm 1, iterates by updating an estimate of the solution support by adding or removing one constraint to/from the active set; then it solves a relaxed QP, obtained by ignoring the non-negativity constraint and restricting to the current support.

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \left\| \boldsymbol{M}_{\bar{\mathcal{Y}}} \boldsymbol{p}_{\bar{\mathcal{Y}}} \right\|_2^2 - \boldsymbol{\eta}^\top \boldsymbol{A}_{\bar{\mathcal{Y}}} \boldsymbol{p}_{\bar{\mathcal{Y}}} \\ \text{w.r.t.} \quad & \boldsymbol{p}_{\bar{\mathcal{Y}}} \in \mathbb{R}^{|\bar{\mathcal{Y}}|} \\ \text{subject to} \quad & \mathbf{1}^\top \boldsymbol{p}_{\bar{\mathcal{Y}}} = 1 \end{aligned} \tag{4.8}$$

whose solution can be found by solving the Karush–Kuhn–Tucker (KKT) system

$$\begin{bmatrix} \boldsymbol{M}_{\bar{\mathcal{Y}}}{}^{\top}\boldsymbol{M}_{\bar{\mathcal{Y}}} & \mathbf{1} \\ \mathbf{1}^{\top} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_{\bar{\mathcal{Y}}} \\ \tau \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{\bar{\mathcal{Y}}}^{\top}\boldsymbol{\eta} \\ 1 \end{bmatrix}. \tag{4.9}$$

At each iteration, the (symmetric) design matrix in Equation 4.9 is updated by adding or removing a row and a column; therefore its inverse (or a decomposition) may be efficiently maintained and updated. The optimal step size for moving a feasible current estimate $\boldsymbol{p}'$ toward a solution $\hat{\boldsymbol{p}}$ of Equation 4.9, while keeping feasibility, is given by (Martins et al., 2015, Equation 31)

$$\gamma = \min\left(1, \min_{y \in \bar{\mathcal{Y}},\ p'(y) > \hat{p}(y)} \frac{p'(y)}{p'(y) - \hat{p}(y)}\right) \tag{4.10}$$

When $\gamma \leq 1$ this update zeros out a coordinate of $\boldsymbol{p}'$; otherwise, $\bar{\mathcal{Y}}$ remains the same. The Wolfe gap at a point $\boldsymbol{d} = [\boldsymbol{d}_u, \boldsymbol{d}_v]$ can be used as a stopping condition and is given by

$$\begin{aligned} \operatorname{gap}(\boldsymbol{d}, \boldsymbol{u}') &\coloneqq \left\langle -\nabla_{\boldsymbol{u}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{d}_u \right\rangle + \left\langle -\nabla_{\boldsymbol{v}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{d}_v \right\rangle \\ &= \left\langle \boldsymbol{\eta}_u - \boldsymbol{u}', \boldsymbol{d}_u \right\rangle + \left\langle \boldsymbol{\eta}_v, \boldsymbol{d}_v \right\rangle. \end{aligned} \tag{4.11}$$

Discussion and comparison. In this presentation, it becomes apparent that the active set method has a very similar structure to the conditional gradient family of algorithms: the key difference being the "approximate refitting" by solving the relaxed KKT system. Both algorithms enjoy global linear convergence with similar rates (Lacoste-Julien and Jaggi, 2015), but the active set algorithm also exhibits exact finite convergence—this allows it, for instance, to capture the optimal sparsity pattern (Nocedal and Wright, 1999, Chapters 16.4 & 16.5). Vinyes and Obozinski (2017) provide a more in-depth discussion of the connections between the two algorithms, also employed for submodular optimization by Bach (2013, Chapter 9).

Algorithm 1 Active Set algorithm for SparseMAP

Initialize: $y^{(0)} \leftarrow \mathsf{MAP}_{\boldsymbol{A}}(\boldsymbol{\eta}_u, \boldsymbol{\eta}_v)$ $\bar{\mathcal{Y}}^{(0)} = \{y^{(0)}\}$; $\boldsymbol{p}^{(0)} = \boldsymbol{e}_{y^{(0)}}$; $[\boldsymbol{u}^{(0)}, \boldsymbol{v}^{(0)}] = \boldsymbol{a}_{y^{(0)}}$
for $t = 0 \dots t_{\mathsf{max}}$ **do**
  Solve the relaxed QP restricted to $\bar{\mathcal{Y}}^{(t)}$; get $\hat{\boldsymbol{p}}, \hat{\tau}, \hat{\boldsymbol{u}} = \boldsymbol{M}\hat{\boldsymbol{p}}$ (Equation 4.9)
  if $\hat{\boldsymbol{p}} = \boldsymbol{p}^{(t)}$ **then**
    $y \leftarrow \mathsf{MAP}_{\boldsymbol{A}}(\boldsymbol{\eta}_u - \hat{\boldsymbol{u}}, \boldsymbol{\eta}_v)$
    if $\mathrm{gap}(\boldsymbol{a}_y, \hat{\boldsymbol{u}}) \leq \hat{\tau}$ **then**
      return $\boldsymbol{u}^{(t)}$ (Equation 4.11)
    else
      $\bar{\mathcal{Y}}^{(t+1)} \leftarrow \bar{\mathcal{Y}}^{(t)} \cup \{y\}$
    end if
  else
    Compute step size γ (Equation 4.10)
    $\boldsymbol{p}^{(t+1)} \leftarrow (1-\gamma)\boldsymbol{p}^{(t)} + \gamma\hat{\boldsymbol{p}}$ (sparse update)
    Update $\bar{\mathcal{Y}}^{(t+1)}$ if necessary
  end if
end for

We perform an empirical comparison on a dependency parsing instance with random potentials. Figure 4.3 shows that active set substantially outperforms all CG variants, both in terms of objective value as well as in the solution sparsity, suggesting that the quadratic curvature makes SparseMAP solvable in very few iterations to high accuracy. We therefore use the active set solver in the remainder of the paper.

4.4 Backward Pass Computation

In order to use SparseMAP as a neural network layer trained with backpropagation, one must compute products of the SparseMAP Jacobian with a vector $\boldsymbol{d}_{\boldsymbol{p}}$. Computing the Jacobian of an optimization problem is an active research topic known as *argmin differentiation*, and is generally difficult. Fortunately, as we show next, argmin differentiation is always easy and efficient in the case of SparseMAP.

Proposition 4.1 *Denote a* SparseMAP *solution by* $\boldsymbol{p}^\star$ *and its support by* $\bar{\mathcal{Y}} := \{y : p^\star(y) > 0\}$. *Then,* SparseMAP *is differentiable almost everywhere with Jacobian*

$$\frac{\partial \boldsymbol{u}^\star}{\partial \boldsymbol{\eta}} = \boldsymbol{M}\boldsymbol{D}(\bar{\mathcal{Y}})\boldsymbol{A}^\top, \text{ where } \boldsymbol{D}(\bar{\mathcal{Y}}) \text{ is a symmetrical matrix defined as}$$

$$\boldsymbol{d}(\bar{\mathcal{Y}})_y := \begin{cases} \left(\boldsymbol{I} - \frac{1}{\mathbf{1}^T\boldsymbol{Z}\mathbf{1}}\boldsymbol{Z}\mathbf{1}\mathbf{1}^T\right)\boldsymbol{z}_y, & y \in \bar{\mathcal{Y}}, \\ \mathbf{0}, & y \notin \bar{\mathcal{Y}}; \end{cases}$$

$$\boldsymbol{Z} := (\boldsymbol{M}_{\bar{\mathcal{Y}}}{}^\top \boldsymbol{M}_{\bar{\mathcal{Y}}})^{-1}.$$

The proof, given in Appendix A.4, relies on the KKT conditions of the SparseMAP QP. Importantly, because $\boldsymbol{D}(\bar{\mathcal{Y}})$ is zero outside of the support of the solution, computing the Jacobian only requires the columns of $\boldsymbol{M}$ and $\boldsymbol{A}$ corresponding to the structures in the active set. Moreover, when using the active set algorithm discussed in Section 4.3, the matrix $\boldsymbol{Z}$ is readily available as a byproduct of the forward pass. The complexity of the backward pass is therefore linear in the number of variables as well as in the size of the support.

Our approach for gradient computation draws its efficiency from the solution sparsity and does not depend on the type of structure considered. This is contrasted with two related lines of research. The first is "unrolling" iterative inference algorithms, for instance belief propagation (Stoyanov et al., 2011) and gradient descent (Belanger et al., 2017), where the backward pass complexity scales with the number of optimization iterations. The second, employed by Kim et al. (2017), when inference can be performed via dynamic programming, backpropagation can be performed using second-order expectation semirings (Li and Eisner, 2009) or more general smoothing (Mensch and Blondel, 2018). The latter approach results in backward pass complexity matching the forward pass. Another noteworthy advantage of our approach is that neither the forward nor the backward passes

involve logarithms, exponentiations or log-domain classes, avoiding the slowdown and stability issues normally incurred.

Connection to sparsemax. Recall that we may recover sparsemax by setting $\boldsymbol{M} = \boldsymbol{I}$. Thus, we also have $\boldsymbol{S} = \boldsymbol{I}$, and the backward pass requires a Jacobian-vector product of the form

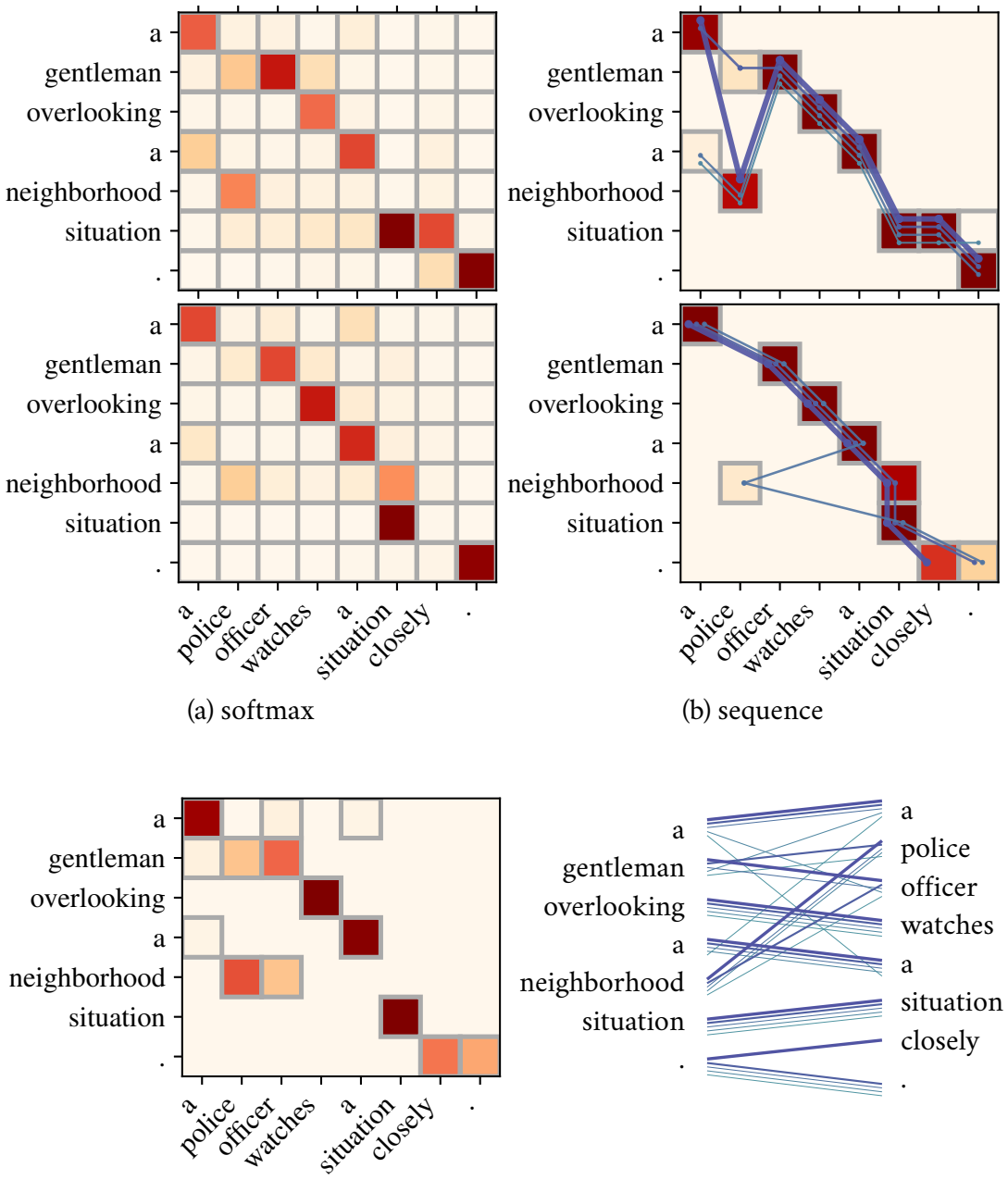
$$\left(\frac{\partial \boldsymbol{u}^\star}{\partial \boldsymbol{\eta}_u}\right)^\top \boldsymbol{d_p} = \boldsymbol{I}_{\bar{\mathcal{Y}}} \left(\boldsymbol{I} - \frac{1}{|\bar{\mathcal{Y}}|}\mathbf{1}\mathbf{1}^\top\right) \boldsymbol{I}_{\bar{\mathcal{Y}}}^\top \boldsymbol{d_p}.$$

This is of course equal to the sparsemax backward pass (Martins and Astudillo, 2016), but is factored in a revealing way, corresponding to a sparsity-aware implementation: (1) selecting the active subset of $\boldsymbol{d_p}$: $(\boldsymbol{d_p})_{\bar{\mathcal{Y}}} = \boldsymbol{I}_{\bar{\mathcal{Y}}}^\top \boldsymbol{d_p}$; (2) computing its mean: $\bar{\boldsymbol{d}}_{\boldsymbol{p}} = \frac{1}{|\bar{\mathcal{Y}}|}\mathbf{1}^\top (\boldsymbol{d_p})_{\bar{\mathcal{Y}}}$; (3) subtracting the mean element-wise: $(\boldsymbol{d_p})_{\bar{\mathcal{Y}}} - \mathbf{1}\bar{\boldsymbol{d}}_{\boldsymbol{p}}$; (4) "scattering" the low dimensional result into the original space, by multiplying by $\boldsymbol{I}_{\bar{\mathcal{Y}}}$. The comparison illustrates how SparseMAP extends to structured problems, takes into account correlations between different structures, as captured by $\boldsymbol{Z}$.

4.5 Sparse Alignment for Natural Language Inference

In this section, we experiment with SparseMAP hidden layers on the natural language inference task: the task of classifying the relationship between a *premise* sentence and a *hypothesis* sentence, previously discussed in Chapter 3.

Baseline. We build upon the state-of-the-art inference model ESIM (Chen et al., 2017), which aligns the words in the *premise* and the *hypothesis* to each other. ESIM is related to the inter-sentence decomposable alignment model of Parikh et al. (2016), mainly differing through the extra LSTM processing layers. Algorithm 2



(a) softmax

(b) sequence

(c) matching

Figure 4.4: Latent alignments on an example from the SNLI validation set, correctly predicted as *neutral* by all compared models. The premise is on the y-axis, the hypothesis on the x-axis. For softmax and matching alignment, on top, columns sum to 1; on the bottom, rows sum to 1. The matching alignment is symmetrical and thus shown only once. Nonzero weights are marked with a border. The structures selected by sequential alignment are overlaid as paths; the selected matchings are displayed in the bottom right.

describes a generic variant of ESIM. The baseline, corresponding to the original model proposed by Chen et al. (2017), implements the align operator as independent softmax over the rows and over the columns of $\boldsymbol{G}$. Following the original authors, pool is implemented as the concatenation of mean-pooling and max-pooling, the MLP has two hidden layers separated by a ReLU nonlinearity, and all hidden layers are 300-dimensional (*i. e.*, the first LSTM is 150-dimensional in each direction, the second is 300-dimensional in each direction).

Algorithm 2 Enhanced Sequential Inference Model (ESIM) with generic alignment

Input: premise $\mathsf{P} = (w_1^{\mathsf{P}}, \ldots, w_{d_{\mathsf{P}}}^{\mathsf{P}})$, hypothesis $\mathsf{H} = (w_1^{\mathsf{H}}, \ldots, w_{d_{\mathsf{H}}}^{\mathsf{H}})$ (lists of words)
Output: predicted class probability $p(c|\mathsf{P}, \mathsf{H})$

1: Look up word embeddings:
$\boldsymbol{h}_i^{\mathsf{P}} \leftarrow \mathsf{lookup}(w_i^{\mathsf{P}}) \in \mathbb{R}^d, \quad \boldsymbol{h}_j^{\mathsf{H}} \leftarrow \mathsf{lookup}(w_i^{\mathsf{H}}) \in \mathbb{R}^d$
2: Encode sequences:
$(\tilde{\boldsymbol{h}}_1^{\mathsf{P}}, \ldots, \tilde{\boldsymbol{h}}_{d_{\mathsf{P}}}^{\mathsf{P}}) \leftarrow \mathsf{BiLSTM}_1(\boldsymbol{h}_1^{\mathsf{P}}, \ldots, \boldsymbol{h}_{d_{\mathsf{P}}}^{\mathsf{P}})$,
$(\tilde{\boldsymbol{h}}_1^{\mathsf{H}}, \ldots, \tilde{\boldsymbol{h}}_{d_{\mathsf{H}}}^{\mathsf{H}}) \leftarrow \mathsf{BiLSTM}_1(\boldsymbol{h}_1^{\mathsf{H}}, \ldots, \boldsymbol{h}_{d_{\mathsf{H}}}^{\mathsf{H}})$
3: Compute alignment scores
$\boldsymbol{G} \in \mathbb{R}^{d_{\mathsf{P}} \times d_{\mathsf{H}}} \quad g_{ij} = \langle \tilde{\boldsymbol{h}}_i^{\mathsf{P}}, \tilde{\boldsymbol{h}}_j^{\mathsf{H}} \rangle$
4: Compute alignment probabilities:
$(\boldsymbol{U}^{\mathsf{P}}, \boldsymbol{U}^{\mathsf{H}}) \leftarrow \mathsf{align}(\boldsymbol{G})$
5: "Augment" each word with the corresponding aligned weighted average:
$\bar{\boldsymbol{h}}_i^{\mathsf{P}} = [\tilde{\boldsymbol{h}}_i^{\mathsf{P}}, \sum_j u_{ij}^{\mathsf{H}} \tilde{\boldsymbol{h}}_j^{\mathsf{H}}]$
$\bar{\boldsymbol{h}}_j^{\mathsf{H}} = [\tilde{\boldsymbol{h}}_j^{\mathsf{H}}, \sum_i u_{ji}^{\mathsf{P}} \tilde{\boldsymbol{h}}_i^{\mathsf{P}}]$
6: Process the augmented vectors through another bidirectional LSTM and pool:
$\boldsymbol{r}^{\mathsf{P}} \leftarrow \mathsf{pool}(\mathsf{BiLSTM}_2(\bar{\boldsymbol{h}}_1^{\mathsf{P}}, \ldots, \bar{\boldsymbol{h}}_{d_{\mathsf{P}}}^{\mathsf{P}}))$,
$\boldsymbol{r}^{\mathsf{H}} \leftarrow \mathsf{pool}(\mathsf{BiLSTM}_2(\bar{\boldsymbol{h}}_1^{\mathsf{H}}, \ldots, \bar{\boldsymbol{h}}_{d_{\mathsf{H}}}^{\mathsf{H}}))$,
$\boldsymbol{r} = [\boldsymbol{r}^{\mathsf{P}}, \boldsymbol{r}^{\mathsf{H}}, \boldsymbol{r}^{\mathsf{P}} - \boldsymbol{r}^{\mathsf{H}}, \boldsymbol{r}^{\mathsf{P}} \odot \boldsymbol{r}^{\mathsf{H}}]$
7: $p(c|\mathsf{P}, \mathsf{H}) \leftarrow \mathsf{softmax}_c(\mathsf{MLP}(\boldsymbol{r}))$

Proposed structured alignment models. We propose using SparseMAP instead, as a *global* alignment mechanism. We consider two types of alignment structures for the align operator in Algorithm 2:

- **Sequential alignment:** We model the alignment of P to H as a sequence tagging instance of length d_{P}, with n possible tags corresponding to the d_{H} words of the hypothesis. Through *transition scores*, we enable the model to capture continuity and monotonicity of alignments: we parametrize transitioning from word t_1 to t_2 by binning the distance $t_2 - t_1$ into 5 groups, $\{-2 \text{ or less}, -1, 0, 1, 2 \text{ or more}\}$. We similarly parametrize the initial alignment using bins $\{1, 2 \text{ or more}\}$ and the final alignment as $\{-2 \text{ or less}, -1\}$, allowing the model to express whether an alignment starts at the beginning or ends on the final word of $\boldsymbol{h}$; formally

$$\eta_u(i, t_1, t_2) \coloneqq \begin{cases} w_{\text{bin}(t_2 - t_1)} & 0 < i < n, \\ w^{\text{start}}_{\text{bin}(t_2)} & i = 0, \\ w^{\text{end}}_{\text{bin}(t_1)} & i = n. \end{cases}$$

  Like the softmax baseline, we align H to P applying the same method in the other direction, with different transition scores $\boldsymbol{w}$. Overall, sequential alignment requires learning 18 additional scalar parameters.

- **Matching alignment:** We now seek a symmetrical alignment in both directions simultaneously. To this end, we cast the alignment problem as finding a maximal weight bipartite matching. We recall from Section 4.1 that a solution can be found via the Hungarian algorithm (in contrast to marginal inference, which is #P-complete). When the premise and the hypothesis have the same number of words, maximal matchings can be represented as permutation matrices; otherwise, we allow some words to remain unaligned (effectively padding the shorter one with infinite-cost words). SparseMAP returns a weighted average of a few maximal matchings. This method requires no additional learned parameters.

ESIM variant	MultiNLI	SNLI
softmax	76.05 (100%)	86.52 (100%)
sequential	75.54 (13%)	**86.62** (19%)
matching	**76.13** (8%)	86.05 (15%)

Table 4.1: Test accuracy scores for natural language inference with structured and unstructured variants of ESIM. In parentheses: the percentage of pairs of words with nonzero alignment scores.

Results. We evaluate the two models alongside the softmax baseline on the SNLI (Bowman et al., 2015) and MultiNLI (Williams et al., 2017) datasets.[2] All models are trained by the stochastic gradient method, with 0.9× learning rate decay at epochs when the validation accuracy is not the best seen. We tune the learning rate on the grid $\{2^k : k \in \{-6, -5, -4, -3\}\}$, extending the range if the best model is at either end. SparseMAP alignments have strong classification performance on both the SNLI and the MultiNLI datasets. The alignments they induce are sparse when seen as attention weights, but can also be represented as sparse convex combinations of a small number of global, discrete structures: this sparsity is not only useful for visualization and interpretability purposes, but also paves the way to a more direct use of discrete structures themselves within a neural network, as we explore in Chapter 5.

Finally, our timing results indicate that structured attention does not constitute a bottleneck in the ESIM model; on the contrary, *matching* alignment can be even slightly faster than the baseline, thanks to sparsity and to the fact that a single computation returns symmetrical weights for alignment in both directions.

[2]We split the MultiNLI matched validation set into equal validation and test sets; for SNLI we use the provided split.

4.6 Discussion and Related Work

Structured attention networks. Structured hidden layers were proposed in the works of Kim et al. (2017) and Liu and Lapata (2018), who take advantage of the tractability of marginal inference in certain structured models and derive specialized backward passes. In contrast, our approach is more general and easier to apply: with SparseMAP, the forward pass only requires MAP inference, and the backward pass is efficiently computed based on the forward pass results. Moreover, unlike marginal inference, SparseMAP yields sparse solutions, which is an appealing property statistically, computationally, and visually.

***K*-best inference.** The support of a SparseMAP solution is a ranked set of likely structures, not unlike K-best inference, often used in pipeline NLP systems for increasing recall and handling uncertainty (Yang and Cardie, 2013). K-best inference can be solved in tree-structured factor graphs (Yanover and Weiss, 2004) and in certain types of structures via specialized algorithms (Camerini et al., 1980; Chegireddy and Hamacher, 1987), as well as approximated in general (Fromer and Globerson, 2009), in time proportional to K calls to MAP inference, although (unlike SparseMAP) not *in terms of* MAP inference. Furthermore, SparseMAP yields a distribution, while K-best does not reveal the posterior gap between structures.

Learning permutations. A popular approach for differentiable permutation learning involves mean-entropic optimal transport relaxations (Adams and Zemel, 2011; Mena et al., 2018). Unlike SparseMAP, this strategy does not apply to general structures, and solutions are not directly expressible as combinations of a few discrete structures (*i. e.*, permutations).

Regularized inference. Ravikumar et al. (2010), Meshi et al. (2015), and Martins et al. (2015) proposed ℓ_2 perturbations and penalties in various related ways, with the goal of solving LP-MAP approximate inference in graphical models. In contrast, the goal of our work is sparse structured prediction, which is not considered in the aforementioned work. Nevertheless, some of the formulations in their work share properties with SparseMAP; exploring the connections further is an interesting avenue for future work.

Herding. An optimization problem closely related to SparseMAP has been employed for the task of *herding* (Bach et al., 2012; Lacoste-Julien et al., 2015), which amounts to approximating a point in a polytope as a sparse combination of vertices. While not directly related to the SparseMAP setting, herding could prove useful for specific structures in which the SparseMAP forward pass enjoys faster specialized algorithms that do not yield a decomposition $\boldsymbol{p}$.

4.7 Chapter Summary

In this chapter, we introduced a new framework for sparse structured inference, SparseMAP, and proposed efficient ways to compute the forward and backward passes of the SparseMAP transform. Experimental results on structured hidden layers demonstrate that SparseMAP leads to strong, interpretable networks trained end-to-end. Modular by design, SparseMAP can be applied readily to any structured problem for which MAP inference is available, including difficult combinatorial problems such as linear assignment.

An interesting future direction is to relax SparseMAP to approximate MAP in-

ference, in order to accommodate more complicated structured problems such as loopy factor graphs; the modular design of SparseMAP is well-suited for this challenge. Another avenue of future work involves semi-supervised structured hidden layers: the SparseMAP loss can readily be applied to hidden layers in end-to-end models, with the cost of a single inference call. Finally, we pointed cases where MAP inference is tractable but marginal inference is difficult, but the reverse case is also possible, for instance in determinantal point processes (Kulesza and Taskar, 2012). Exploring backpropagation through inference in such models is an interesting direction for future work.

CHAPTER 5

DYNAMICALLY INFERRED NEURAL NETWORK STRUCTURE

In the previous chapter, we introduced SparseMAP, a differentiable sparse inference strategy allowing hidden layers with structured outputs. The output of such a hidden layer, however, is just a vector encoding of the selected structures. We now extend this idea one step further, by permitting the latent structure to define arbitrary structure-dependent computation.

A host of neural architectures for NLP tasks are designed to follow some underlying structural representation of the data itself; for instance, tree-structured recursive neural networks compose words according to the syntactic relationships between them, typically using off-the-shelf parsers. Recent attempts to jointly learn the latent structure encounter a trade-off: make factorization assumptions that limit expressiveness, or sacrifice end-to-end differentiability. Using the recently proposed SparseMAP inference, which retrieves a sparse distribution over latent structures, we propose a novel approach for end-to-end learning of latent structure predictors jointly with a downstream predictor. To the best of our knowledge, our method is the first to enable unrestricted dynamic computation graph construction from the *global* latent structure, while maintaining differentiability.

5.1 Overview and Related Work

Latent structure models are a powerful tool for modeling compositional data and building NLP pipelines (Smith, 2011). An interesting emerging direction is to **dynamically** adapt a network's computation graph, based on structure inferred from the input; notable applications include learning to write programs (Bosnjak

et al., 2017), answering visual questions by composing specialized modules (Hu et al., 2017; Johnson et al., 2017), speeding up image classification by skipping hidden layers (Veit and Belongie, 2017), and composing sentence representations using latent syntactic parse trees (Yogatama et al., 2017).

But how to learn a model that is able to condition on such combinatorial variables? How to marginalize over latent structures? For tractability, existing approaches have to make a choice. Some of them eschew *global* latent structure, resorting to computation graphs built from smaller local decisions: *e. g.*, structured attention networks (Kim et al., 2017; Liu and Lapata, 2018), as well as our networks from Chapter 4, use local posterior marginals $\boldsymbol{u}$ as attention weights, Veit and Belongie (2017) make binary decisions to keep or skip each layer, Maillard et al. (2017) construct sentence representations from parser chart entries. Others allow more flexibility at the cost of losing end-to-end differentiability, ending up with reinforcement learning problems (Yogatama et al., 2017; Hu et al., 2017; Johnson et al., 2017; Williams et al., 2018). Approaches based on continuous relaxations of discrete structures, such as iterative neural decoders (Martins and Kreutzer, 2017), have similar shortcomings.

More traditional approaches employ an off-line structure predictor (*e. g.*, a parser) to define the computation graph (Tai et al., 2015; Chen et al., 2017), sometimes with some parameter sharing (Bowman et al., 2016). However, these off-line methods are unable to *jointly* train the latent model and the downstream classifier via error gradient information.

We propose here a new strategy for building **dynamic computation graphs** with latent structure, through **sparse structure prediction**. Sparsity allows selecting and conditioning on a tractable number of global structures, eliminating

the limitations stated above. Namely, our approach is the first that:

A) is **fully differentiable**;

B) supports **latent structured variables**;

C) can marginalize over full **global structures**.

This contrasts with off-line and with reinforcement learning-based approaches, which satisfy **B** and **C** but not **A**; and with local marginal-based methods such as structured attention networks, which satisfy **A** and **B**, but not **C**. Key to our approach is **SparseMAP inference**, introduced in Chapter 4, which induces, for each data example, a very sparse posterior distribution over the possible structures, allowing us to compute the expected network output efficiently and explicitly in terms of a small, interpretable set of latent structures. Our model can be trained end-to-end with gradient-based methods, without the need for policy exploration or sampling.

We demonstrate our strategy on inducing latent dependency TreeLSTMs, achieving competitive results on sentence classification, natural language inference, and reverse dictionary lookup.

5.2 Models with Latent Structured Variables

We describe our proposed approach for training latent variable models, where the latent variables are combinatorial structures, in particular non-projective dependency parse trees.

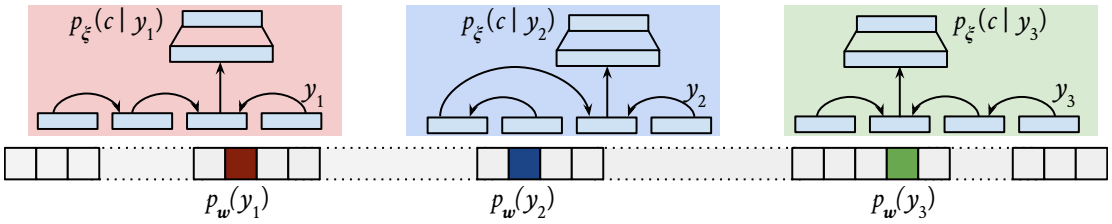


Figure 5.1: Our method computes a sparse probability distribution over all possible latent structures: in this illustration, there are only three dependency trees with nonzero probability. For each such structure y, we may evaluate $p_\xi(c \mid x, y)$ by constructing the corresponding computation graph. For conciseness, the dependence on x is omitted from the figure.

Let x denote the input, and c denote an output class. Denote by $y \in \mathcal{Y}(x)$ a latent structured variable; for example, $\mathcal{Y}(x)$ denotes the set of possible dependency trees for x. We would like to train a neural network to model

$$p(c \mid x) := \sum_{y \in \mathcal{Y}(x)} p_w(y \mid x)\, p_\xi(c \mid x, y), \tag{5.1}$$

where $p_w(y \mid x)$ is a structured-output parsing model that defines a distribution over trees, and $p_\xi(c \mid x, y)$ is a classifier whose computation graph may depend **freely and globally** on the structure y (*e. g.*, a TreeLSTM). The rest of this section focuses on the challenge of defining $p_w(y \mid x)$ such that Equation 5.1 remain tractable and differentiable.

Global inference. Denote by $\theta_w(y;\, x)$ a scoring function, assigning each tree a non-normalized score. For instance, we may have an *arc-factored* score $\theta_w(y; x) := \sum_{a \in y} \eta_w(a; x)$, where we interpret a tree y as a set of directed arcs a, each receiving an atomic score $\eta_w(a; x)$. As discussed in Section 4.1, deriving p_w given θ_w is known as *structured inference*. This can be written as a Ω-regularized optimization problem of the form

$$p_w(\cdot|x) := \operatorname*{arg\,max}_{q \in \triangle^{|\mathcal{Y}(x)|}} \sum_{y \in \mathcal{Y}(x)} q(y)\theta_w(y; x) - \Omega(q), \tag{5.2}$$

where $\triangle^{|\mathcal{Y}(x)|}$ is the set of all possible probability distributions over $\mathcal{Y}(x)$, and Ω is a convex regularizer. Note that this time we are interested in the entire probability distribution, not just the variable marginals, as in Chapter 4.

Marginal inference, obtained by setting $\Omega(\boldsymbol{q}) \coloneqq \sum_{y \in \mathcal{Y}(x)} q(y) \log q(y)$, provides a differentiable expression for $p_{\boldsymbol{w}}$. However, crucially, since $\exp(\cdot) > 0$, every tree is assigned strictly nonzero probability. Therefore—unless the downstream p_ξ is constrained to also factor over arcs, as in Kim et al. (2017); Liu and Lapata (2018), or more generally in Mensch and Blondel (2018)—the sum in Equation 5.1 requires enumerating the exponentially large $\mathcal{Y}(x)$. This is generally intractable, and even hard to approximate via sampling, even when $p_{\boldsymbol{w}}$ is tractable. At the polar opposite, setting $\Omega(\boldsymbol{q}) \coloneqq 0$ yields **maximum a posteriori (MAP) inference**, which assigns a probability of 1 to the highest-scoring tree, and 0 to all others, yielding a very sparse $p_{\boldsymbol{w}}$. However, since the top-scoring tree (or top-k, for fixed k) does not vary with small changes in $\boldsymbol{w}$, error gradients cannot propagate through MAP. This prevents end-to-end gradient-based training for MAP-based latent variables, which makes them more difficult to use.

SparseMAP inference. Our key insight is to use the sparse posterior induced by SparseMAP (Chapter 4) to sparsify the set $\mathcal{Y}$ while preserving differentiability. SparseMAP uses a quadratic penalty on the posterior marginals

$$\Omega(\boldsymbol{q}) \coloneqq \|\boldsymbol{u}(\boldsymbol{q})\|_2^2, \quad \text{where} \quad [\boldsymbol{u}(\boldsymbol{q})]_a \coloneqq \sum_{h:a \in h} q(h).$$

SparseMAP assigns nonzero probability to only a small set of plausible trees $\bar{\mathcal{Y}} \subset \mathcal{Y}$, of size at most equal to the number of arcs (Martins et al., 2015, Proposition 11). This guarantees that the summation in Equation 5.1 can be computed efficiently by iterating over $\bar{\mathcal{Y}}$, as depicted in Figure 5.1. The algorithms from Section 4.3

can be used to compute $p_{\boldsymbol{w}}$ in the forward pass. [1] It remains to characterize the **backward pass**, *i. e.*, $\partial p_{\boldsymbol{w}}(y \mid x)/\partial \boldsymbol{w}$, in order to enable end-to-end gradient-based training of structured latent variable models. The next proposition provides this.

Proposition 5.1 *Let $p_{\boldsymbol{w}}(y|x)$ denote the* SparseMAP *posterior probability distribution, i. e., a solution of Equation 5.2 for $\Omega(\boldsymbol{q}) = \|\boldsymbol{u}(\boldsymbol{q})\|_2^2$, where $u_a(\boldsymbol{q}) = \sum_{y:a\in y} q(y) = \sum_y m_{a,y} q(y)$ for an appropriately defined indicator matrix $\boldsymbol{M}$. Define $\boldsymbol{Z} \coloneqq \left(\boldsymbol{M}_{\bar{\mathcal{Y}}(x)}{}^\top \boldsymbol{M}_{\bar{\mathcal{Y}}(x)}\right)^{-1} \in \mathbb{R}^{|\bar{\mathcal{Y}}|\times|\bar{\mathcal{Y}}|}$, the sum of column y of $\boldsymbol{Z}$ by $\varsigma(y) \coloneqq \sum_{y'\in\bar{\mathcal{Y}}(x)} z_{y',y}$, and the overall sum of $\boldsymbol{Z}$ by $\zeta \coloneqq \sum_{y'\in\bar{\mathcal{Y}}(x)} \varsigma(y')$.*

Then, for any $y \in \mathcal{Y}(x)$, we have

$$\frac{\partial p_{\boldsymbol{w}}(y|x)}{\partial \boldsymbol{w}} = \begin{cases} \sum_{y'\in\bar{\mathcal{Y}}(x)} \left(z_{y,y'} - \zeta^{-1}\varsigma(y)\varsigma(y')\right) \partial\theta_{\boldsymbol{w}}(y';\, x)/\partial \boldsymbol{w}, & p_{\boldsymbol{w}}(y|x) > 0 \\ 0, & p_{\boldsymbol{w}}(y|x) = 0. \end{cases}$$

Crucially, this gradient term has the same sparsity pattern as $p_{\boldsymbol{w}}$, and is efficient to compute, amounting to multiplying by a $|\bar{\mathcal{Y}}(x)|$-by-$|\bar{\mathcal{Y}}(x)|$ matrix. The proof is given in Appendix A.5 as a slight variation of the SparseMAP backward pass (Proposition 4.1).

Generality. Our description focuses on probabilistic classifiers, but our method can be readily applied to networks that output any representation, not necessarily a class probability. For this, we define a function $\boldsymbol{r}_{\boldsymbol{\xi}}(x, y)$, consisting of any auto-differentiable computation w.r.t. x, conditioned on the discrete latent structure y in arbitrary, non-differentiable ways. We then compute

$$\bar{\boldsymbol{r}}(x) \coloneqq \sum_{y\in\mathcal{Y}(x)} p_{\boldsymbol{w}}(y \mid x)\boldsymbol{r}_{\boldsymbol{\xi}}(x, y) = \mathbb{E}_{y\sim p_{\boldsymbol{w}}}\boldsymbol{r}_{\boldsymbol{\xi}}(x, y).$$

[1] While the SparseMAP arc posteriors $\boldsymbol{u}$ are always unique, for pathologic inputs (*i. e.*, not in general position), there may be more than one optimal distribution $p_{\boldsymbol{w}}$. This did not pose any problems in practice, where any ties would be broken at random.

This strategy is demonstrated in our reverse-dictionary experiments. In addition, our approach is not limited to trees: any structured model with tractable MAP inference may be used.

5.3 Latent Dependency TreeLSTM

We combine the word vectors $\boldsymbol{v}_i$ in a sentence into a single vector using a tree-structured Child-Sum LSTM, which allows an arbitrary number of children at any node (Tai et al., 2015). Our baselines consist in extreme cases of dependency trees: where the parent of word i is word $i + 1$ (resulting in a **left-to-right** sequential LSTM), and where all words are direct children of the root node (resulting in a **flat** additive model). We also consider **off-line** dependency trees precomputed by Stanford CoreNLP (Manning et al., 2014).

Neural arc-factored dependency parsing. To compute the arc score $\eta_{\boldsymbol{w}}(a; x)$ we follow Kiperwasser and Goldberg (2016) in using a multi-layer perceptron with one hidden layer.

5.4 Experiments

Setup and overview. We tackle three natural language processing tasks: sentence-level classification (on sentiment and subjectivity datasets), natural language inference, and reverse dictionary lookup. In all cases, we use the common abstraction of a *sentence encoder*: a module that composes a sentence into a single vector. All networks are trained via the stochastic gradient method with 16 samples

per batch. We tune the learning rate on the validation set on a logarithmic grid. We decay the learning rate by a factor of 0.9 after every epoch at which the validation performance is not the best seen, and stop after five epochs without improvement. At test time, we scale the arc scores η_w by a temperature t chosen on the validation set, controlling the sparsity of the `SparseMAP` distribution. All hidden layers are 300-dimensional.

Sentence classification. We evaluate our models for sentence-level subjectivity classification (Pang and Lee, 2004) and for binary sentiment classification on the Stanford Sentiment Treebank (Socher et al., 2013). In both cases, we use a softmax output layer on top of the Dependency TreeLSTM output representation.

Natural language inference. We apply our strategy to the SNLI corpus (Bowman et al., 2015), which consists of classifying premise-hypothesis sentence pairs into entailment, contradiction or neutral relations. In this case, for each pair (x_P, x_H), the running sum is over *two* latent distributions over parse trees:

$$\sum_{\substack{y_P \in \mathcal{Y}(x_P) \\ y_H \in \mathcal{Y}(x_H)}} p_\xi(c \mid x_{\{P,H\}}, y_{\{P,H\}})\; p_w(y_P|x_P)\; p_w(y_H|x_H).$$

For each pair of trees, we independently encode the premise and hypothesis using a dependency TreeLSTM. To obtain a pair encoding, we concatenate the two vectors, their difference, and their element-wise product (Mou et al., 2016). The result is passed through one *tanh* hidden layer, followed by the softmax output layer.

Reverse dictionary lookup. The reverse dictionary task aims to compose the words in a dictionary definition into an embedding that is close to the defined

	subj.	SST	SNLI
left-to-right	**92.71**	82.10	80.98
flat	92.56	83.96	81.74
off-line	92.15	83.25	81.37
latent	92.25	**84.73**	**81.87**

Table 5.1: Test accuracy for classification and natural language inference.

word. We therefore used *fixed* input and output embeddings, set to unit-norm 500-dimensional vectors provided, together with training and evaluation data, by Hill et al. (2016). For this task, the output is a computed by projecting the TreeLSTM encoding back to the dimension of the word embeddings, and normalizing the result to have unit ℓ_2 norm. The training objective is to maximize the cosine similarity of the predicted embedding with the word being defined. The evaluation is grouped into seen definitions, unseen definitions, and conceptual definitions.

Baselines. We compare our latent TreeLSTM to fixed-dependency baselines:

- **left-to-right**: every sentence is composed sequentially from left to right: this is essentially a standard LSTM recurrent encoder;
- **flat**: sentences are considered flat trees, where all words are directly connected to the root. This ignores all order information and is similar to an *deep averaging network*, except for the LSTM-style gated composition rule;
- **off-line**: we obtain predicted dependency trees from Stanford CoreNLP.

Results. Classification and NLI results are reported in Table 5.1. Compared to the latent structure model of Yogatama et al. (2017), our model performs better on SNLI (80.5%) but worse on SST (86.5%). On SNLI, our model also outperforms

	seen			unseen			concepts		
	rank	acc^{10}	acc^{100}	rank	acc^{10}	acc^{100}	rank	acc^{10}	acc^{100}
left-to-right	17	42.6	73.8	43	33.2	61.8	28	35.9	66.7
flat	18	45.1	71.1	**31**	**38.2**	**65.6**	29	34.3	68.2
latent	**12**	**47.5**	**74.6**	40	35.6	60.1	**20**	**38.4**	**70.7**
Maillard et al. (2017)	58	30.9	56.1	40	33.4	57.1	40	57.1	62.6
Hill et al. (2016)	12	48	28	22	41	70	69	28	54

Table 5.2: Results on the reverse dictionary lookup task (Hill et al., 2016). Following the authors, for an input definition, we rank a shortlist of approximately 50k candidate words according to the cosine similarity to the output vector, and report median rank of the expected word, accuracy at 10, and at 100.

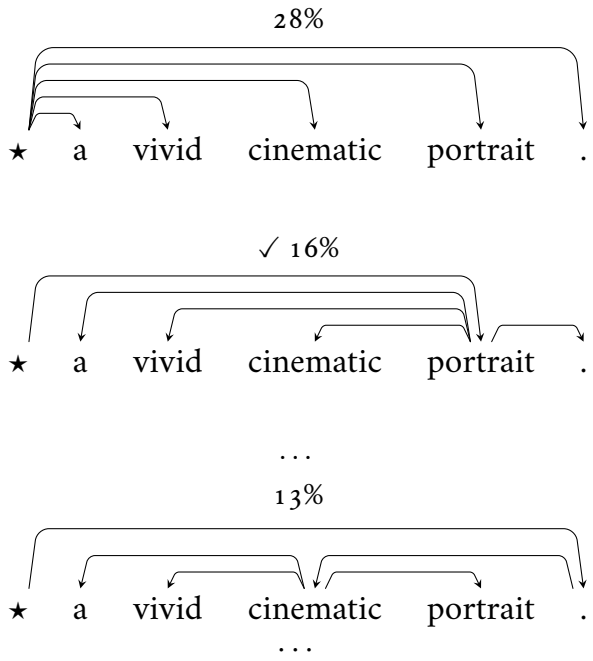


Figure 5.2: Three of the sixteen trees with nonzero probability for an SST test example. Flat trees, such as the first one, perform well on this task, as reflected by the baselines. The second tree, marked with ✓, agrees with the off-line parser.

Maillard et al. (2017) (81.6%). To our knowledge, latent structure models have not been tested on subjectivity classification. Surprisingly, the simple flat and left-to-right baselines are very strong, outperforming the off-line dependency tree models on all three datasets. The latent dependency tree model reaches the best accuracy on two out of the three datasets. Its performance is, in part, due to its

adaptability: Figure 5.2 shows that, on sentiment classification, where the flat baseline is strong, the latent model prefers flat parses, while also assigning high scores to meaningful deeper ones. On reverse dictionary lookup (Table 5.2), our latent model also performs well, most noticeably on the concept classification task, where the input definitions are more different from the ones seen during training. For comparison, we repeat here the CKY-based latent TreeLSTM model of Maillard et al. (2017), as well as the LSTM-based model of Hill et al. (2016); as these results are from different-sized models, they are not entirely comparable.

5.5 Chapter Summary

We presented a novel approach for training latent structure neural models, based on the key idea of sparsifying the set of possible structures. We used the proposed method to train competitive models based on latent dependency TreeLSTMs. The flexibility of our method opens up several avenues for future work.

On one side, our method can be used with any structure for which `MAP` inference is available (*e. g.*, matchings, alignments). On another, since we have no restrictions on the way in which $p_\xi(y, h)$ may depend on the latent structure h, we may design latent versions of more complicated state-of-the-art models, such as ESIM for natural language inference (Chen et al., 2017). Partial or distant supervision (Yogatama et al., 2017), can be applied to the latent parsers in our models for little computational cost, using the `SparseMAP` loss introduced in the next chapter.

CHAPTER 6

SPARSEMAP LOSSES FOR SPARSE STRUCTURED PREDICTION

Up to this point, our focus has been on models of latent structure inside neural networks: algorithms that can automatically infer structure in the data without direct supervision. We now turn our attention to supervised **structured output prediction**, where we explicitly train machine learning models to predict desired structures. We show that SparseMAP inference proves useful in this scenario as well, through a combination of interpretability and effective handling of uncertainty.

The key piece necessary for using SparseMAP for structured prediction is an appropriate *structured loss*, analogous to the commonly used Structured SVM or the CRF losses. To derive such a SparseMAP loss, we first build a generic family of losses for classification and structured prediction, which we call Fenchel-Young losses, based on well-known fundamental results from convex analysis. Many established structured and unstructured losses are part of this family, confirming the relevance of our proposed SparseMAP loss.

This chapter is based on part of (Niculae et al., 2018).

6.1 Structured Prediction Losses

When training machine learning models to predict structured outputs, it is common, just like in the unstructured case, to minimize a loss function that calibrates the predictions with the ground truth. Because of the combinatorially large number of possible structures, the choice of structured loss functions is slightly more limited than in the unstructured case. In this section, we review the most commonly used structured prediction losses.

In the sequel, we assume, as in Chapter 4, that all possible structures are assigned a score θ_y, parametrized as $\boldsymbol{\theta} = \boldsymbol{A}^\top \boldsymbol{\eta}$, where $\boldsymbol{\eta} \coloneqq \boldsymbol{\eta}_{\boldsymbol{w}}(x)$ is generated by some model whose weights $\boldsymbol{w}$ we aim to learn. We denote by $\bar{y} \in \mathcal{Y}$ the desired gold-label structure. Learning a structured-output model over a dataset of N samples can then be seen as an optimization problem of the form

$$\arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} L(\boldsymbol{\eta}_{\boldsymbol{w}}(x_i), \bar{y}_i). \tag{6.1}$$

Maximum likelihood and the CRF loss. Recall from Section 4.1 that we can use an exponential family to define the probability of any structure y, induced by the scores $\boldsymbol{\theta}$ (Equation 4.1)

$$p(y) = \frac{\exp \theta(y)}{\sum_{y' \in \mathcal{Y}} \exp \theta(y')} = \frac{\exp \boldsymbol{a}_y^\top \boldsymbol{\eta}}{\sum_{y' \in \mathcal{Y}} \exp \boldsymbol{a}_{y'}^\top \boldsymbol{\eta}}$$

This naturally suggests that a sensible training objective is to maximize the probability of the correct structure, or, equivalently, to minimize the *log-likelihood*

$$L(\boldsymbol{\eta}, \bar{y}) = -\boldsymbol{a}_{\bar{y}}^\top \boldsymbol{\eta} + \log \sum_{y \in \mathcal{Y}} \exp \boldsymbol{a}_y^\top \boldsymbol{\eta} \tag{6.2}$$

This structured loss is known as the Conditional Random Field (CRF) loss (Lafferty et al., 2001). The term $\log \sum_{y \in \mathcal{Y}} \exp \boldsymbol{a}_y^\top \boldsymbol{\eta} \coloneqq \log Z$ is often known as the log-partition function, or the *cumulant*, and its computation is the main difficulty posed by training with the CRF loss. The cumulant can be connected to marginal inference, if we observe that

$$\nabla_{\boldsymbol{\eta}} L(\boldsymbol{\eta}, \bar{y}) = -\boldsymbol{a}_{\bar{y}} + \mathbb{E}_{Y \sim \boldsymbol{p}}[\boldsymbol{a}_Y]. \tag{6.3}$$

Indeed, typically, specialized algorithms for marginal inference (*e. g.*, the forward-backward algorithm) also compute $\log Z$ along the way.

Training with the CRF loss thus requires marginal inference, therefore, as discussed in Chapter 4, no exact general algorithms exist. While approximate algo-

rithms exist, a lot of attention has instead focused on learning with MAP inference, for which general approximations have been understood more firmly via linear programming theory. The following two structured losses illustrate this.

Structured perceptron. A simple but effective way for training structured models is the structured perceptron (Collins, 2002), an extension of the perceptron algorithm dating back to Rosenblatt (1958). It corresponds to minimizing the loss

$$L(\boldsymbol{\eta}, \bar{y}) = -\boldsymbol{a}_{\bar{y}}^{\top}\boldsymbol{\eta} + \max_{y \in \mathcal{Y}} \boldsymbol{a}_{y}^{\top}\boldsymbol{\eta} \tag{6.4}$$

The perceptron loss is not differentiable but it is subdifferentiable. If $\widehat{y}$ is a structure achieving the maximum above, then by Danskin's theorem (Danskin, 1966)

$$\boldsymbol{a}_{\widehat{y}} - \boldsymbol{a}_{\bar{y}} \in \partial L(\boldsymbol{\eta}, \bar{y}). \tag{6.5}$$

Therefore, subgradient learning can be performed as long as we have access to MAP inference.

Structured hinge. Emerging from a line of work involving structured version of Support Vector Machines (Taskar et al., 2003; Tsochantaridis et al., 2004, 2005), the structured hinge loss can be seen as a perceptron loss augmented with a cost term

$$L(\boldsymbol{\eta}, \bar{y}) = -\boldsymbol{a}_{\bar{y}}^{\top}\boldsymbol{\eta} + \max_{y \in \mathcal{Y}} \left(\boldsymbol{a}_{y}^{\top}\boldsymbol{\eta} + \rho(y, \bar{y})\right) \tag{6.6}$$

where ρ is a user-specified cost of predicting structure y instead of $\bar{y}$. If the cost can decompose into the parts defined by $\boldsymbol{A}$, *i. e.*, $\rho(y, \bar{y}) = \boldsymbol{a}_{y}^{\top}\boldsymbol{\rho}_{\bar{y}}$, then the max in Equation 6.6 can be rearranged as

$$\max_{y \in \mathcal{Y}} \boldsymbol{a}_{y}^{\top}(\boldsymbol{\eta} + \boldsymbol{\rho}_{\bar{y}}),$$

which is known as *cost-augmented* MAP *inference* and can be computed by running a MAP inference algorithm on the modified scores $\boldsymbol{\eta}' = \boldsymbol{\eta} + \boldsymbol{\rho}_{\bar{y}}$. As above, by Danskin's rule, a subgradient is given by

$$\boldsymbol{a}_{\widehat{y}} - \boldsymbol{a}_{\bar{y}} \in \partial L(\boldsymbol{\eta}, \bar{y}). \tag{6.7}$$

except this time $\widehat{y}$ is obtained by solving the cost-augmented inference problem rather than the traditional MAP problem.

Interpolating between the above. While less used in practice, it has been noted that the cost augmentation technique is not specifically dependent on MAP inference, and a similar cost-augmented CRF loss can be obtained (Gimpel and Smith, 2010). In fact, a family of continuous interpolations between the losses discussed above is given by Martins et al. (2010) as

$$L_{\beta,\gamma}(\boldsymbol{\eta}, \bar{y}) = \frac{1}{\beta} \log \sum_{y \in \mathcal{Y}} \exp\left[\beta \boldsymbol{\eta}^\top (\boldsymbol{a}_y - \boldsymbol{a}_{\bar{y}}) + \gamma \rho(y, \bar{y})\right] \tag{6.8}$$

We further note that, in the unstructured case when $\boldsymbol{A} = \boldsymbol{I}$, the structured losses above become corresponding multi-class classification losses. For example, the structured hinge loss, with an appropriately chosen ρ, reduces to the classification hinge loss in Equation 2.4.

6.2 A Fenchel-Young Family of Losses

In this section, we introduce a general family of losses that will make the newly proposed SparseMAP loss arise as a very natural case. Below, we let $\Omega : \mathbb{R}^D \to \mathbb{R}$

denote a convex penalty function and denote by $\Omega_\triangle$ its restriction to $\triangle^D \subset \mathbb{R}^D$, *i. e.*,

$$\Omega_\triangle(\boldsymbol{p}) \coloneqq \begin{cases} \Omega(\boldsymbol{p}), & \boldsymbol{p} \in \triangle^D; \\ \infty, & \boldsymbol{p} \notin \triangle^D. \end{cases}$$

The Fenchel convex conjugate of $\Omega_\triangle$ (Equation 2.6) is

$$\Omega^*_\triangle(\boldsymbol{\theta}) \coloneqq \sup_{\boldsymbol{p} \in \mathbb{R}^D} \boldsymbol{\theta}^\top \boldsymbol{p} - \Omega_\triangle(\boldsymbol{p}) = \sup_{\boldsymbol{p} \in \triangle^D} \boldsymbol{\theta}^\top \boldsymbol{p} - \Omega(\boldsymbol{p}).$$

We next introduce a family of generic loss functions.

Definition 6.1 *Let $\Omega : \mathbb{R}^d \to \mathbb{R}$ be a convex penalty function. The **Fenchel-Young loss** $L_\Omega : \mathbb{R}^d \times \triangle^d \to \mathbb{R}_+$ is*

$$L_\Omega(\boldsymbol{\theta}, \bar{\boldsymbol{p}}) \coloneqq \Omega^*_\triangle(\boldsymbol{\theta}) + \Omega_\triangle(\bar{\boldsymbol{p}}) - \boldsymbol{\theta}^\top \boldsymbol{p}. \tag{6.9}$$

Definition 6.2 *Let $\Omega : \mathbb{R}^D \to \mathbb{R}$ be a convex penalty function, $\boldsymbol{A}$ be a $(k \times D)$-dimensional matrix encoding the structure of a problem, $\bar{\boldsymbol{p}} \in \triangle^D$ be a ground-truth target distribution over structures, and $\boldsymbol{\eta} \in \mathbb{R}^d$ be a vector of predicted loss potential. The **structured Fenchel-Young loss** $L^{\boldsymbol{A}}_\Omega : \mathbb{R}^d \times \triangle^D \to \mathbb{R}_+$ is*

$$L^{\boldsymbol{A}}_\Omega(\boldsymbol{\eta}, \bar{\boldsymbol{p}}) \coloneqq \Omega^*_\triangle(\boldsymbol{A}^\top \boldsymbol{\eta}) + \Omega_\triangle(\bar{\boldsymbol{p}}) - \boldsymbol{\eta}^\top \boldsymbol{A} \bar{\boldsymbol{p}}. \tag{6.10}$$

As the name suggests, this family in rooted in the Fenchel-Young duality gap (Equation 2.7). A more in-depth study of Fenchel-Young losses, out of scope of this text, is given in Blondel et al. (2018). In the sequel, we only focus on **structured** Fenchel-Young losses.

When the ground-truth target is peaked on a single gold structure, *i. e.*, $\bar{\boldsymbol{p}} = \boldsymbol{e}_{\bar{y}}$, for careful choices of Ω we uncover the well known losses discussed in Section 6.1.

- CRF: $\Omega \equiv -H$;
- Structured perceptron: $\Omega \equiv 0$;
- Structured SVM: $\Omega \equiv \rho(\cdot, \bar{\boldsymbol{p}})$ for a cost function ρ;
- Margin CRF: $\Omega \equiv -H + \rho(\cdot, \bar{\boldsymbol{p}})$.

The next proposition states properties of structured Fenchel-Young losses, including a general connection between a loss and its corresponding inference method, thus providing a constructive way of deriving new structured losses.

Proposition 6.1 *Let $\Omega, \boldsymbol{A}, \boldsymbol{\eta}, \bar{\boldsymbol{p}}$ as in Definition 6.2. Denote the* ***generalized inference*** *output $\widehat{\boldsymbol{p}_\Omega}(\boldsymbol{\eta}) \in \partial\Omega^*_\triangle(\boldsymbol{\eta}) = \operatorname{arg\,max}_{\boldsymbol{p} \in \triangle^D} \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} - \Omega_\triangle(\boldsymbol{p})$. Then, the following properties hold:*

1. ***Non-negativity.*** *$L^{\boldsymbol{A}}_\Omega(\boldsymbol{\eta}, \bar{\boldsymbol{p}}) \geq 0$.*
2. ***Zero loss.*** *The loss is zero iff $\bar{\boldsymbol{p}} \in \partial\Omega^*_\triangle(\boldsymbol{\eta})$, i. e., iff $\bar{\boldsymbol{p}} \in \operatorname{arg\,max}_{\boldsymbol{p} \in \triangle^D} \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} - \Omega_\triangle(\boldsymbol{p})$;*
3. ***Convexity & gradient.*** *$L^{\boldsymbol{A}}_\Omega(\boldsymbol{\eta}, \bar{\boldsymbol{p}})$ is convex, $\partial L^{\boldsymbol{A}}_\Omega(\boldsymbol{\eta}, \bar{\boldsymbol{p}}) \ni \boldsymbol{A}(\widehat{\boldsymbol{p}_\Omega} - \bar{\boldsymbol{p}})$;*
4. ***Temperature scaling.*** *For any $t \in \mathbb{R}_+$, i. e., $t > 0$, $L^{\boldsymbol{A}}_{t\Omega}(\boldsymbol{\eta}, \bar{\boldsymbol{p}}) = tL^{\boldsymbol{A}}_\Omega(\boldsymbol{\eta}/t, \bar{\boldsymbol{p}})$ and $\widehat{\boldsymbol{p}_{t\Omega}}(\boldsymbol{\eta}) \in \partial\Omega^*_\triangle(\boldsymbol{\eta}/t)$.*

Proof is given in Appendix A.6. The non-negativity and zero loss properties suggests that training a model to minimize $L^{\boldsymbol{A}}_\Omega$ calibrates the model toward predicting the true label. Property 3 shows how to compute a loss subgradient provided access to some inference output $[\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}] \in \boldsymbol{A}\widehat{\boldsymbol{p}_\Omega} \in \mathbb{R}^d$.

The temperature scaling property suggests that the strength of the penalty Ω can be adjusted by simply scaling $\boldsymbol{\eta}$. In consequence, the structured Fenchel-Young losses also generalize the interpolated loss family $L_{\beta,\gamma}$ in Equation 6.8, since

$$L_{\beta,\gamma}(\boldsymbol{\eta}, \bar{\boldsymbol{p}}) = \frac{1}{\beta} L^{\boldsymbol{A}}_\Omega(\beta\boldsymbol{\eta}, \bar{\boldsymbol{p}})$$

for $\Omega = -H + \gamma\beta\rho(\cdot, \bar{\boldsymbol{p}})$. As before, setting $\boldsymbol{A} = \boldsymbol{I}$ leads to a natural reduction to the unstructured case.

In the same way in which CRFs are equivalent to maximum entropy classifiers, Fenchel-Young losses correspond to a generalized maximum entropy principle, with respect to the generalized entropy induced by Ω (Blondel et al., 2018).

SparseMAP losses. The Fenchel-Young loss framework provides a natural way of defining SparseMAP losses, by applying the appropriate convex penalty. Specifically, we may define the following losses.

Definition 6.3 (SparseMAP losses.) *We define the following* SparseMAP *losses as* $L_\Omega^{\boldsymbol{A}}$*, where* $\boldsymbol{A} = [\boldsymbol{M}, \boldsymbol{N}]$ *is a matrix encoding the structure, with the following choices of* Ω*:*

- SparseMAP*:* $\Omega(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{M}\boldsymbol{p}\|_2^2$,
- *Margin* SparseMAP*:* $\Omega(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{M}\boldsymbol{p}\|_2^2 + \rho(\boldsymbol{p}, \bar{\boldsymbol{p}})$,

where ρ *is a cost as in the structured hinge loss case (Equation 6.6).*

The properties of Fenchel-Young losses ensure that the SparseMAP losses indeed encourage models to make predictions close to the desired ground truth. As seen in Section 6.1, computing subgradients of the structured perceptron / structured SVM losses requires MAP inference on the marginal polytope $\mathcal{M}_{\boldsymbol{A}}$, while the CRF loss gradient requires marginal inference. Similarly, the subgradients of the SparseMAP loss can be computed via SparseMAP inference, for which we have proposed an efficient algorithm in Section 4.2, relying only on MAP oracles. This result makes SparseMAP losses computationally promising for structured prediction.

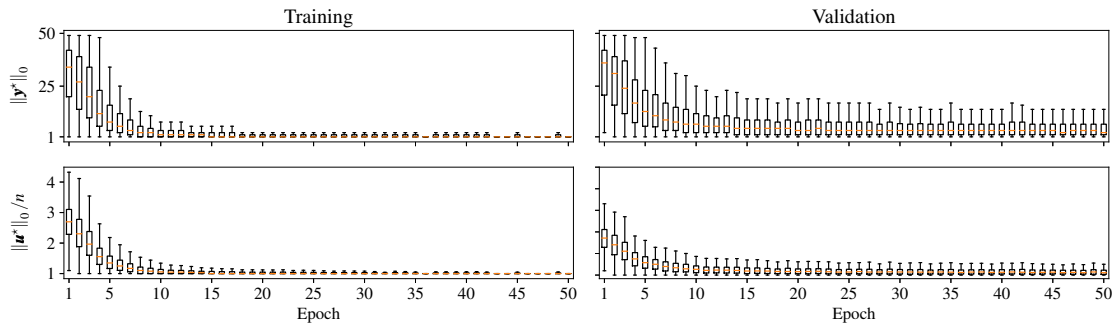


Figure 6.1: Distribution of the tree sparsity (top) and arc sparsity (bottom) of SparseMAP solutions during training on the Chinese dataset. Shown are respectively the number of trees and the average number of parents with non-zero probability per word.

Loss	en	zh	vi	ro	ja
Structured SVM	87.02	81.94	69.42	87.58	**96.24**
CRF	86.74	83.18	69.10	87.13	96.09
SparseMAP	86.90	**84.03**	69.71	87.35	96.04
m-SparseMAP	**87.34**	82.63	**70.87**	**87.63**	96.03
UDPipe baseline	87.68	82.14	69.63	87.36	95.94

Table 6.1: Unlabeled attachment accuracy scores for dependency parsing, using a bi-LSTM model (Kiperwasser and Goldberg, 2016). SparseMAP and m-SparseMAP yield the best parser on 4/5 datasets. For context, we include the scores of the CoNLL 2017 UDPipe baseline, trained under the same conditions (Straka and Straková, 2017).

6.3 Sparse Dependency Parsing with SparseMAP Losses

We evaluate the SparseMAP losses against the commonly used CRF and structured SVM losses. The task we focus on is non-projective *dependency parsing*: a structured output task consisting of predicting the directed tree of grammatical dependencies between words in a sentence (Jurafsky and Martin, 2018, Chapter 14). We use annotated Universal Dependency data (Nivre et al., 2016), as used in the CoNLL 2017 shared task (Zeman et al., 2017). To isolate the effect of the loss, we use the provided gold tokenization and part-of-speech tags. We follow closely the bidirectional LSTM arc-factored parser of Kiperwasser and Goldberg (2016), using the same

model configuration: 100-dimensional word embeddings, 25-dimensional part-of-speech embeddings, 100-dimensional hidden multi-layer perceptron (MLP) layers, two 125-dimensional bi-LSTM hidden layers, and a word dropout probability of .25; the only difference is not using externally pretrained word embeddings, since the quality of available word embeddings is different across languages. Parameters are trained using Adam (Kingma and Ba, 2015), tuning the learning rate on the grid $\{.5, 1, 2, 4, 8\} \times 10^{-3}$, expanded by a factor of 2 if the best model is at either end.

We experiment with 5 languages, diverse both in terms of family and in terms of the amount of training data (ranging from 1,400 sentences for Vietnamese to 12,525 for English). Test set results (Table 6.1) indicate that the SparseMAP losses outperform the SVM and CRF losses on 4 out of the 5 languages considered. This suggests that SparseMAP is a good middle ground between MAP-based and marginal-based losses in terms of smoothness and gradient sparsity.[1]

Moreover, as illustrated in Figure 6.1, the SparseMAP loss encourages **sparse predictions**: models converge towards sparser solutions as they train, yielding very few ambiguous arcs. When confident, SparseMAP can predict a single tree. Otherwise, the small set of candidate parses returned can be easily visualized, often indicating genuine linguistic ambiguities, as exemplified in Figure 6.2. This property of SparseMAP is valuable in pipeline systems, *e. g.*, when the output of a dependency parser is the input to a downstream application: error propagation is diminished in cases where the highest-scoring tree is incorrect (which is the case for the sentences in Figure 6.2). Unlike K-best heuristics, SparseMAP dynamically adjusts its output sparsity, which is desirable on realistic data where most instances are easy.

[1]Results for the perceptron loss are not included as it consistently performed drastically worse than all others.

We evaluate the SparseMAP losses against the commonly used CRF and structured SVM losses. The task we focus on is non-projective *dependency parsing*: a structured output task consisting of predicting the directed tree of grammatical dependencies between words in a sentence (Jurafsky and Martin, 2018, Chapter 14). We use annotated Universal Dependency data (Nivre et al., 2016). To isolate the effect of the loss, we use the provided gold tokenization and part-of-speech tags. We follow closely the bidirectional LSTM arc-factored parser of Kiperwasser and Goldberg (2016), using the same model configuration, and experiment with a set of 5 languages, diverse both in terms of language family as well as the amount of training data.

Baselines. We compare the SparseMAP loss, and its margin variant, with the well-established Structured SVM and CRF losses.

Results. SparseMAP losses outperform the SVM and CRF losses on 4 out of the 5 languages considered, suggesting a good middle ground between MAP-based and marginal-based losses in terms of smoothness and gradient sparsity. Moreover, we showcase that the SparseMAP loss encourages **sparse predictions**: models converge towards sparser solutions as they train, resulting in very few ambiguous arcs. When confident, SparseMAP can predict a single tree. Otherwise, the small set of candidate parses returned can be easily visualized, often indicating genuine linguistic ambiguities. This property of SparseMAP is valuable in pipeline systems, *e. g.*, when the output of a dependency parser is the input to a downstream application: error propagation is diminished in cases where the highest-scoring tree is incorrect. Unlike K-best heuristics, SparseMAP dynamically adjusts its output sparsity, which is desirable on realistic data where most instances are easy.

6.4 Related Work

Proper scoring rules, *a.k.a.* proper losses, (Grünwald and Dawid, 2004; Gneiting and Raftery, 2007; Williamson et al., 2016) are a key tool in the context of probability forecasting (Dawid, 1986) to express the goodness-of-fit (lower is better) between a ground truth $\bar{\boldsymbol{p}} \in \mathcal{Y}$ and a vector of predictions $\boldsymbol{\theta} \in \triangle^{|\mathcal{Y}|}$.[2] (In this formulation, $\boldsymbol{\theta}$ must be in the simplex.) The fact that any scoring rule induces a generalized entropy is well-known (DeGroot, 1962; Grünwald and Dawid, 2004). The reverse mapping has also been studied. Then we can construct, from a generalized negative entropy Ω, a scoring rule $S_\Omega \colon \triangle^d \times \mathcal{Y} \to \mathbb{R}$ as follows (Savage, 1971; Gneiting and Raftery, 2007; Reid et al., 2015)

$$S_\Omega(\boldsymbol{\theta}; \bar{\boldsymbol{p}}) \coloneqq \langle \nabla\Omega(\boldsymbol{\theta}), \bar{\boldsymbol{p}} - \boldsymbol{\theta} \rangle - \Omega(\boldsymbol{\theta}) = B_\Omega(\bar{\boldsymbol{p}} || \boldsymbol{\theta}) - \Omega(\bar{\boldsymbol{p}}), \tag{6.11}$$

recovering the well-known relation between scoring rules and Bregman divergences (Gneiting and Raftery, 2007; Williamson et al., 2016). As an example, choosing the Gini index $\mathsf{H}(\boldsymbol{p}) = 1 - \|\boldsymbol{p}\|^2$ and $\Omega = -\mathsf{H}$ generates the **Brier score** (Brier, 1950) $S_\Omega(\boldsymbol{\theta}; \bar{\boldsymbol{p}}) = \sum_{\boldsymbol{p} \in \mathcal{Y}} \left(\imath[\bar{\boldsymbol{p}} = \boldsymbol{p}] - \boldsymbol{\theta}^\top \bar{\boldsymbol{p}}\right)^2$. Thus sparsemax and the Brier score share the same generating function. A crucial difference between S_Ω and L_Ω, however, is that S_Ω is **not necessarily convex** in its first argument (Williamson et al. (2016, Proposition 17) shows that it is in fact quasi-convex) while L_Ω **always is**. In addition, the first argument is **constrained** to $\triangle^{|\mathcal{Y}|}$ for S_Ω, while it is **unconstrained** for L_Ω. For this reason, S_Ω is usually composed with an **invertible** link function (Buja et al., 2005; Williamson et al., 2016), while this is not necessary with L_Ω. Finally, a classification loss construction was recently proposed by Duchi et al. (2018, Proposition 3). Fenchel-Young losses are broader, supporting e.g. regression, and in fact include (Duchi et al., 2018, Proposition 3) as a special case.

[2] For ease of notation, in this section we assume $\mathcal{Y} = \{\boldsymbol{e}_i\}_{i=1}^{|\mathcal{Y}|}$. This is the case in the cited work, although our formulation naturally extends to distributions.

Smoothing techniques were used extensively in Shalev-Shwartz and Zhang (2016) to create smoothed losses. However, these techniques were applied on a per-loss basis, while we propose a generic loss construction, with clear links between smoothing or regularization and the probability distribution over classes induced by $\nabla\Omega^*$.

6.5 Chapter Summary

We develop a generic family of loss functions, covering both structured and unstructured prediction, underscoring many well-known losses such as the logistic loss and the hinge loss. We show that Fenchel-Young losses have many desirable properties and make it easy to construct new losses, in particular our new `SparseMAP` losses for structured prediction. The `SparseMAP` losses lead to strong models that make sparse, interpretable predictions, a good fit for tasks where local ambiguities are common, like many natural language processing tasks.

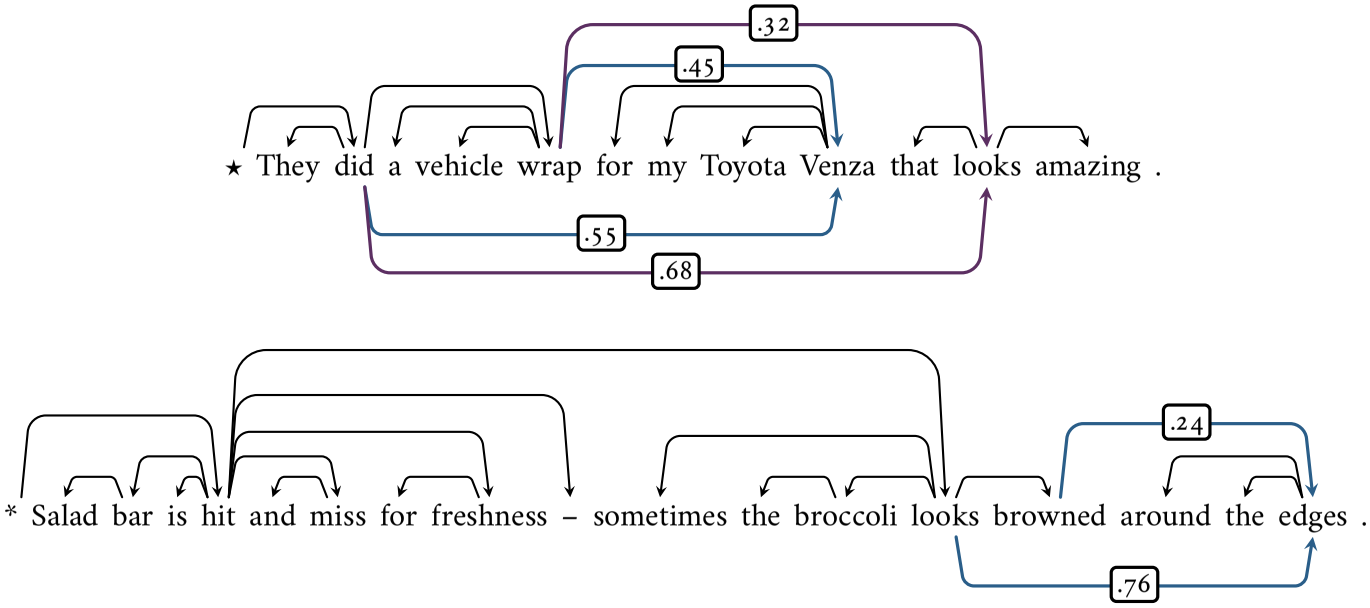


Figure 6.2: Example of ambiguous parses from the UD English validation set. SparseMAP selects a small number of candidate parses (left: three, right: two), differing from each other in a small number of ambiguous dependency arcs. In both cases, the desired gold parse is among the selected trees (depicted by the arcs above the sentence), but it is not the highest-scoring one.

CHAPTER 7

MINING ARGUMENT STRUCTURES WITH EXPRESSIVE FACTOR GRAPHS

With our focus still on predicting structured outputs, we now focus on a specific application: **argument mining**. We design an expressive *factor graph* using domain insights, leading to superior performance.

Our novel factor graph model for argument mining is designed for settings in which the argumentative relations in a document do not necessarily form a tree structure. (This is the case in over 20% of the web comments dataset we mainly study.) Our model jointly learns elementary unit type classification and argumentative relation prediction. Moreover, our model supports both linear (SVM) as well as neural (RNN) parametrizations, can enforce structure constraints (*e. g.*, transitivity), and can express dependencies between adjacent relations and propositions. Our approaches outperform unstructured baselines in both web comments and argumentative essay datasets.

This chapter is based on (Niculae et al., 2017).

7.1 Argumentation and Argument Mining

Argument mining consists of the automatic identification of argumentative structures in documents, a valuable task with applications in policy making, summarization, and education, among others. The argument mining task includes the tightly-knit subproblems of classifying propositions into elementary unit types and detecting argumentative relations between the elementary units. The desired output is a document argumentation graph structure, such as the one in Figure 7.1, where propositions are denoted by letter subscripts, and the associated argumenta-

tion graph shows their types and support relations between them.

Most annotation and prediction efforts in argument mining have focused on tree or forest structures (Peldszus and Stede, 2015; Stab and Gurevych, 2017), constraining argument structures to form one or more trees. This makes the problem computationally easier by enabling the use of maximum spanning tree–style parsing approaches. However, argumentation *in the wild* can be less well-formed. The argument put forth in Figure 7.1, for instance, consists of two components: a simple tree structure and a more complex graph structure (c jointly supports b and d). In this work, we design a flexible and highly expressive structured prediction model for argument mining, jointly learning to classify elementary units (henceforth *propositions*) and to identify the argumentative relations between them (henceforth *links*). By formulating argument mining as inference in a factor graph (Kschischang et al., 2001), our model (described in Section 7.4) can account for correlations between the two tasks, can consider second order link structures (e.g., in Figure 7.1, $c \rightarrow b \rightarrow a$), and can impose arbitrary constraints (e.g., transitivity).

To parametrize our models, we evaluate two alternative directions: linear structured SVMs (Tsochantaridis et al., 2005), and recurrent neural networks with structured loss, extending the model of Kiperwasser and Goldberg (2016). Interestingly, RNNs perform poorly when trained with classification losses, but become competitive with the feature-engineered structured SVMs when trained within our proposed structured learning model.

We evaluate our approach on two argument mining datasets. Firstly, on our new *Cornell eRulemaking Corpus – CDCP*,[1] consisting of argument annotations on comments from an eRulemaking discussion forum, where links don't always form

[1]Dataset available at `http://joonsuk.org`.

[Calling a debtor at work is counter-intuitive;]$_a$ [if collectors are continuously calling someone at work, other employees may report it to the debtor's supervisor.]$_b$ [Most companies have established rules about receiving or making personal calls during working hours.]$_c$ [If a collector or creditor calls a debtor on his/her cell phone and is informed that the debtor is at work, the call should be terminated.]$_d$ [No calls to employers should be allowed,]$_e$ [as this jeopardizes the debtor's job.]$_f$

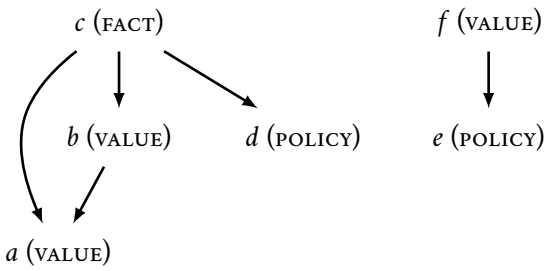


Figure 7.1: Example annotated comment from the CDCP dataset. The proposition types (*e. g.*, FACT, VALUE) will be described in detail in the next sections.

trees (Figure 7.1 shows an abridged example comment, and Section 7.2 describes the dataset in more detail). Secondly, on the UKP argumentative essays v2 (henceforth UKP), where argument graphs are annotated strictly as multiple trees (Stab and Gurevych, 2017). In both cases, the results presented in Section 7.6 confirm that our models outperform unstructured baselines. On UKP, we improve link prediction over the best reported result in (Stab and Gurevych, 2017), which is based on integer linear programming postprocessing. For insight into the strengths and weaknesses of the proposed models, as well as into the differences between SVM and RNN parametrizations, we perform an error analysis in Section 7.7. To support argument mining research, we also release our Python implementation, `Marseille`.[2]

[2]Available at `https://github.com/vene/marseille`.

7.2 Argumentation Data

We release a new argument mining dataset consisting of user comments about rule proposals regarding Consumer Debt Collection Practices (CDCP) by the Consumer Financial Protection Bureau collected from an eRulemaking website, http://regulationroom.org. In this section, we summarize the properties of this dataset. An in-depth description can be found in Park and Cardie (2018).

Policymaking is a central aspect of U.S. governing, and argumentation plays a key part in its process. The eRulemaking initiative brings additional transparency and visibility; the associated websites provide researchers access to intrinsically motivated argumentative commentary (Park et al., 2012). The public comments can actually influence policy rulings (Hochschild and Danielson, 1998), as many agencies are required to address all pertinent questions posed through forums such as the studied one (Lubbers, 2006).

Annotation scheme. Our choice of types of elementary units (*policy, value, fact, testimony*, and *reference*) and support relations (*reason* and *evidence*) is based on the argumentation model proposed by Park et al. (2015a), specifically designed for argumentation structures found in web discussion forums, such as the eRulemaking one we use. Such structures can be more free-form than the ones encountered in controlled, elicited writing such as Peldszus and Stede (2015), making our model, which supports but unrestricted directed graphs, well suited. Indeed, over 20% of the comments in our dataset exhibit local structures that would not be allowable in a tree. Possible link types are *reason* and *evidence*, and proposition types are split into five fine-grained categories: POLICY and VALUE contain subjective judgements/interpretations, where only the former specifies a specific course of action

to be taken. On the other hand, TESTIMONY and FACT do not contain subjective expressions, the former being about personal experience, or "anecdotal." Lastly, REFERENCE covers URLs and citations, which are used to point to objective evidence in an online setting.

In comparison, the UKP dataset (Stab and Gurevych, 2017) only makes the syntactic distinction between CLAIM, MAJOR CLAIM, and PREMISE types, but it also includes *attack* links. The permissible link structure is stricter in UKP, with links constrained in annotation to form one or more disjoint directed trees within each paragraph. Also, since web arguments are not necessarily fully developed, our dataset has many argumentative propositions that are not in any argumentation relations. In fact, it isn't unusual for comments to have no argumentative links at all: 28% of CDCP comments have no links, unlike UKP, where all essays have complete argument structures. Such comments with no links make the problem harder, emphasizing the importance of capturing the *lack* of argumentative support, not only its presence.

Annotation results. Each user comment was annotated by two annotators, who independently annotated the boundaries and types of propositions, as well as the links among them. To produce the final corpus, a third annotator manually resolved the conflicts,[3] and two automatic preprocessing steps were applied: we take the link transitive closure, and we remove a small number of nested propositions.[4] The resulting dataset contains 731 comments, consisting of about 3800 sentences

[3]Inter-annotator agreement is measured with Krippendorf's α (Krippendorff, 1980) with respect to elementary unit type (α=64.8%) and links (α=44.1%). A separate paper describing the dataset is under preparation.

[4]When two propositions overlap, we keep the one that results in losing the fewest links. For generality, we release the dataset without this preprocessing, and include code to reproduce it; we believe that handling nested argumentative units is an important direction for further research.

(≈4700 propositions) and 88k words. Out of the 43k possible pairs of propositions, links are present between only 1300 (roughly 3%). In comparison, UKP has fewer documents (402), but they are longer, with a total of 7100 sentences (6100 propositions) and 147k words. Since UKP links only occur within the same paragraph and propositions not connected to the argument are removed in a preprocessing step, link prediction is less imbalanced in UKP, with 3800 pairs of propositions being linked out of a total of 22k (17%). We reserve a test set of 150 documents (973 propositions, 272 links) from CDCP, and use the provided 80-document test split from UKP (1266 propositions, 809 links).

7.3 Related Work

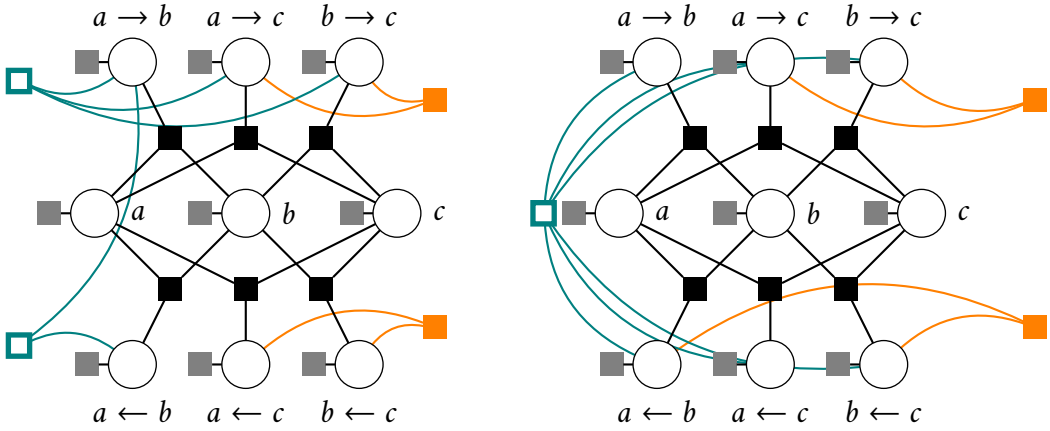
Our factor graph formulation draws from ideas previously used independently in parsing and argument mining. In particular, maximum spanning tree (MST) methods for arc-factored dependency parsing have been successfully used by McDonald et al. (2005b) and applied to argument mining with mixed results by Peldszus and Stede (2015). As they are not designed for the task, MST parsers cannot directly handle proposition classification or model the correlation between proposition and link prediction—a limitation our model addresses. Using RNN features in an MST parser with a structured loss was proposed by Kiperwasser and Goldberg (2016); their model can be seen as a particular case of our factor graph approach, limited to link prediction with a tree structure constraint. Our models support multi-task learning for proposition classification, parametrizing adjacent links with higher-order structures (e.g., $c \rightarrow b \rightarrow a$) and enforcing arbitrary constraints on the link structure, not limited to trees. Such higher-order constrained structures have been successfully used for dependency and semantic parsing (Martins et al., 2013;

Martins and Almeida, 2014; Das et al., 2014); to our knowledge we are the first to apply them to argument mining and to parametrize such factor graphs with neural networks. Stab and Gurevych (2017) used an integer linear program to combine the output of independent proposition and link classifiers using a hand-crafted scoring formula, an approach similar to our baseline. Our factor graph method can combine the two tasks in a more principled way, as it fully learns the correlation between the two tasks without relying on hand-crafted scoring, and therefore can readily be applied to other argumentation datasets. Furthermore, our model can enforce the tree structure constraint, required on the UKP dataset, using MST inference rather than the exponential number of cycle constraints used by Stab and Gurevych (2017), thanks to the AD^3 inference algorithm (Martins et al., 2015).

Sequence tagging has been applied to the related structured tasks of proposition identification and classification (Stab and Gurevych, 2017; Habernal and Gurevych, 2016; Park et al., 2015b); integrating such models is an important next step. Meanwhile, a new direction in argument mining explores *pointer networks* (Potash et al., 2016); a promising method, currently lacking support for tree structures and domain-specific constraints.

7.4 Factor Graphs for Argument Structure Prediction

Binary and multi-class classification have been applied with some success to proposition and link prediction separately, but we seek a way to jointly learn the argument mining problem at the *document* level, to better model contextual dependencies and constraints. We therefore turn to *structured learning*, a framework that provides the desired level of expressiveness.



(a) CDCP (b) UKP

Figure 7.2: Factor graphs for a document with three propositions (a, b, c) and the six possible edges between them, and some of the factors used, illustrating differences and similarities between our models for the two datasets. Unary factors are light grey; compatibility factors are black. Factors not part of the basic model have curved edges: higher-order factors are orange and on the right; link structure factors are hollow, as that they don't have any parameters. Strict constraint factors are omitted for simplicity.

For argument mining, this document-level *factor graph* representation allows us to handle variable document lengths, to capture dependencies between proposition and link predictions, and to restrict valid outputs, (e.g., disallowing link cycles).

Model description. An input document is a string of words with proposition offsets delimited. We denote the propositions in a document by $\{a, b, c, ...\}$ and the possible directed link between a and b as $a \rightarrow b$. The argument structure we seek to predict consists of the type of each proposition $y_a \in \mathcal{P}$, and a binary label for each link $y_{a \rightarrow b} \in \mathcal{R} = \{\texttt{on}, \texttt{off}\}$.[5] The possible proposition types $\mathcal{P}$ differ for the two datasets; such differences are documented in Table 7.1. As we describe the variables and factors constituting a document's factor graph, we shall refer to Figure 7.2 each time for illustration.

[5]For simplicity and comparability, we follow Stab and Gurevych (2017) in using binary link labels; our models are easily extended to "labeled link" factors.

Unary potentials. Each proposition a and each link $a \to b$ has a corresponding random variable in the factor graph (the circles in Figure 7.2). To encode the model's belief in each possible value for these variables, we parametrize the *unary factors* (gray boxes in Figure 7.2) with unary potentials: $\boldsymbol{\phi}(a) \in \mathbb{R}^{|\mathcal{P}|}$ is a score of y_a for each possible proposition type. Similarly, link unary potentials $\boldsymbol{\phi}(a \to b) \in \mathbb{R}^{|\mathcal{R}|}$ are scores for $y_{a \to b}$ being on/off. Without any other factors, this would amount to independent classifiers for each task.

Compatibility factors. For every possible link $a \to b$, the variables $(a, b, a \to b)$ are bound by a dense factor scoring their joint assignment (the black boxes in Figure 7.2). Such a factor could automatically learn to encourage links from compatible types (e.g., from TESTIMONY to POLICY) or discourage links between less compatible ones (e.g., from FACT to TESTIMONY). In the simplest form, this factor would be parametrized as a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{|\mathcal{P}| \times |\mathcal{P}| \times |\mathcal{R}|}$, with t_{ijk} retaining the score of a source proposition of type i to be (k = on) or not to be (k = off) in a link with a proposition of type j. For more flexibility, we parametrize this factor with **compatibility features** depending only on simple structure: $\boldsymbol{t}_{ijk}$ becomes a vector, and the score of configuration (i, j, k) is given by $\boldsymbol{v}_{ab}^{\top} \boldsymbol{t}_{ijk}$ where $\boldsymbol{v}_{ab}$ consists of three binary features:

- **bias:** a constant value of 1, allowing $\boldsymbol{\mathcal{T}}$ to learn a base score for a label configuration (i, j, k), as in the simple form above,
- **adjacency:** when there are no other propositions between the source and the target,
- **order:** when the source precedes the target.

Second order factors. Local argumentation graph structures such as $a \rightarrow b \rightarrow c$ might be modeled better together rather than through separate link factors for $a \rightarrow b$ and $b \rightarrow c$. As in higher-order structured models for semantic and dependency parsing (Martins et al., 2013; Martins and Almeida, 2014), we implement three types of second order factors:

- **grandparent**: $a \rightarrow b \rightarrow c$
- **sibling**: $a \leftarrow b \rightarrow c$.
- **coparent**: $a \rightarrow b \leftarrow c$

Not all of these types of factors make sense on all datasets: as sibling structures cannot exist in directed trees, we don't use sibling factors on UKP. On CDCP, by transitivity, every grandparent structure implies a corresponding sibling, so it is sufficient to parametrize siblings. This difference between datasets is emphasized in Figure 7.2, where one example of each type of factor is pictured on the right side of the graphs (orange boxes with curved edges): on CDCP we illustrate a co-parent factor (top right) and a sibling factor (bottom right), while on UKP we show a co-parent factor (top right) and a grandparent factor (bottom right). We call these factors *second order* because they involve two link variables, scoring the joint assignment of both links being on.

Valid link structure. The global structure of argument links can be further constrained using domain knowledge. We implement this using constraint factors; these have no parameters and are denoted by empty boxes in Figure 7.2. In general, well-formed arguments should be cycle-free. In the UKP dataset, links form a directed forest and can never cross paragraphs. This particular constraint can be

Model part	CDCP dataset	UKP dataset
proposition types	REFERENCE $\succ$ TESTIMONY $\succ$ FACT $\succ$ VALUE $\succ$ POLICY	CLAIM, MAJOR CLAIM, PREMISE
links	all possible	within each paragraph
2^{nd} order factors	siblings, co-parents	grandparents, co-parents
link structure	transitive acyclic: • $a \to b \;\&\; b \to c \implies a \to c$ • AtMostOne($a \to b, b \to a$)	directed forest: • TreeFactor over each paragraph • zero-potential "root" links $a \to *$
strict constraints	link source must be as least as objective as the target: $a \to b \implies a \geq b$	link source must be premise: $a \to b \implies a = \text{PREMISE}$

Table 7.1: Instantiation of model design choices for each dataset.

expressed as a series of *tree factors*,[6] one for each paragraph (the factor connected to all link variables in Figure 7.2). In CDCP, links do not form a tree, but we use logic constraints to enforce transitivity (top left factor in Figure 7.2) and to prevent symmetry (bottom left); the logic formulas implemented by these factors are described in Table 7.1. Together, the two constraints have the desirable side effect of preventing cycles.

Strict constraints. We may include further domain-specific constraints into the model, to express certain disallowed configurations. For instance, proposition types that appear in CDCP data can be ordered by the level of objectivity (Park et al., 2015a), as shown in Table 7.1. In a well-formed argument, we would want to see links from more objective to equally or less objective propositions: it's fine to provide FACT as reason for VALUE, but not the other way around. While the training data sometimes violates this constraint, enforcing it might provide a useful inductive bias.

Inference. Computing the most likely assignment involves a MAP over a factor graph with cycles and many overlapping factors, including logic factors. While exact inference methods are generally unavailable, our setting is perfectly suited for the Alternating Directions Dual Decomposition (AD^3) algorithm: approximate inference on expressive factor graphs with overlapping factors, logic constraints, and generic factors (e.g., directed tree factors) defined through maximization oracles (Martins et al., 2015). When AD^3 returns an integral solution, it is globally optimal, but when solutions are fractional, several options are available. At test time, for

[6] A tree factor regards each bound variable as an edge in a graph and assigns $-\infty$ scores to configurations that are not valid trees. For inference, we can use maximum spanning arborescence algorithms such as Chu-Liu/Edmonds, as discussed in Section 4.1.

analysis, we retrieve exact solutions using the branch-and-bound method. At training time, however, fractional solutions can be used *as-is*; this makes better use of each iteration and actually increases the ratio of integral solutions in future iterations, as well as at test time, as proven by Meshi et al. (2016). We also find that after around 15 training iterations with fractional solutions, over 99% of inference calls are integral.

Learning. We train the models by minimizing the structured hinge loss (Equation 6.6), with a weighted Hamming cost:

$$\rho(\bar{y}, \hat{y}) := \sum_{v} \rho(\bar{y}_v)\, \imath[\bar{y}_v = \hat{y}_v]$$

where v is summed over all variables in a document $\{a\} \cup \{a \rightarrow b\}$, and $\rho(\bar{y}_v)$ is a misclassification cost. We assign uniform costs ρ to 1 for all mistakes except false-negative links, where we use higher cost proportional to the class imbalance in the training split, effectively giving more weight to positive links during training.

7.5 RNN and Linear Parametrizations.

Linear Structured SVM. One option for parametrizing the potentials of the unary and higher-order factors is with *linear models*, using proposition, link, and higher-order features. This gives birth to a linear structured SVM (Tsochantaridis et al., 2005), which, when using l_2 regularization, can be trained efficiently in the dual using the online block-coordinate Frank-Wolfe algorithm of Lacoste-Julien et al. (2013), as implemented in the `pystruct` library (Müller and Behnke, 2014). This algorithm is more convenient than subgradient methods, as it does not require tuning a learning rate parameter.

Features. For unary proposition and link features, we faithfully follow Stab and Gurevych (2017, Tables 9 and 10): proposition features are lexical (unigrams and dependency tuples), structural (token statistics and proposition location), indicators (from hand-crafted lexicons), contextual, syntactic (subclauses, depth, tense, modal, and POS), probability, discourse (Lin et al., 2014), and average GloVe embeddings (Pennington et al., 2014). Link features are lexical (unigrams), syntactic (POS and productions), structural (token statistics, proposition statistics and location features), hand-crafted indicators, discourse triples, PMI, and shared noun counts.

Our proposed higher-order factors for grandparent, co-parent, and sibling structures require features extracted from a proposition triplet a, b, c. In dependency and semantic parsing, higher-order factors capture relationships between words, so sparse indicator features can be efficiently used. In our case, since propositions consist of many words, BOW features may be too noisy and too dense; so for simplicity we again take a cue from the link-specific features used by Stab and Gurevych (2017). Our higher-order factor features are: same sentence indicators (for all 3 and for each pair), proposition order (one for each of the 6 possible orderings), Jaccard similarity (between all 3 and between each pair), presence of any shared nouns (between all 3 and between each pair), and shared noun ratios: nouns shared by all 3 divided by total nouns in each proposition and each pair, and shared nouns between each pair with respect to each proposition. Up to vocabulary size difference, our total feature dimensionality is approximately 7000 for propositions and 2100 for links. The number of second order features is 35.

Argument structure RNN. Neural network methods have proven effective for natural language problems even with minimal-to-no feature engineering. Inspired by the use of LSTMs (Hochreiter and Schmidhuber, 1997) for MST dependency

parsing by Kiperwasser and Goldberg (2016), we parametrize the potentials in our factor graph with an LSTM-based neural network,[7] replacing maximum-spanning tree inference with the more general AD^3 algorithm, and using relaxed solutions for training when inference is inexact.

We extract embeddings of all words with a corpus frequency > 1, initialized with GloVe word vectors. We use a deep bidirectional LSTM to encode contextual information, representing a proposition a as the average of the LSTM outputs of its words, henceforth denoted $\overleftrightarrow{\boldsymbol{a}}$.

Proposition potentials. We apply a multi-layer perceptron (MLP) with rectified linear activations to each proposition, with all layer dimensions equal except the final output layer, which has size $|\mathcal{P}|$ and is not passed through any nonlinearities.

Link potentials. To score a dependency $a \rightarrow b$, Kiperwasser and Goldberg (2016) pass the concatenation $[\overleftrightarrow{\boldsymbol{a}}; \overleftrightarrow{\boldsymbol{b}}]$ through an MLP. After trying this, we found slightly better performance by first passing *each* proposition through a slot-specific dense layer $(\overline{\boldsymbol{a}} := \sigma_{\text{src}}(\overleftrightarrow{\boldsymbol{a}}), \overline{\boldsymbol{b}} := \sigma_{\text{trg}}(\overleftrightarrow{\boldsymbol{b}}))$ followed by a *bilinear* transformation:

$$\phi_{\text{on}}(a \rightarrow b) := \overline{\boldsymbol{a}}^{\top} \boldsymbol{W} \overline{\boldsymbol{b}} + \boldsymbol{w}_{\text{src}}^{\top} \overline{\boldsymbol{a}} + \boldsymbol{w}_{\text{trg}}^{\top} \overline{\boldsymbol{b}} + w_0^{(\text{on})}.$$

Since the bilinear expression returns a scalar, but the link potentials must have a value for both the `on` and `off` states, we set the full potential to $\boldsymbol{\phi}(a \rightarrow b) := [\phi_{\text{on}}(a \rightarrow b), w_0^{(\text{off})}]$ where $w_0^{(\text{off})}$ is a learned scalar bias. We initialize $\boldsymbol{W}$ to the diagonal identity matrix.

[7] We use the `dynet` library (Neubig et al., 2017).

Second order potentials. Grandparent potentials $\phi(a \rightarrow b \rightarrow c)$ score two adjacent directed edges, in other words three propositions. We again first pass each proposition representation through a slot-specific dense layer. We implement a multilinear scorer analogously to the link potentials:

$$\phi(a \rightarrow b \rightarrow c) := \sum_{i,j,k} \bar{\boldsymbol{a}}_i \bar{\boldsymbol{b}}_j \bar{\boldsymbol{c}}_k w_{ijk}$$

where $\mathcal{W} = (w)_{ijk}$ is a third-order cube tensor. To reduce the large numbers of parameters, we implicitly represent $\mathcal{W}$ as a rank r tensor: $w_{ijk} = \sum_{s=1}^{r} u_{is}^{(1)} u_{js}^{(2)} u_{ks}^{(3)}$. Notably, this model captures only third-order interactions between the representation of the three propositions. To capture first-order "bias" terms, we could include slot-specific linear terms, e.g., $\boldsymbol{w}_a^\top \bar{\boldsymbol{a}}$; but to further capture quadratic *backoff* effects (for instance, if two propositions carry a strong signal of being siblings regardless of their parent), we would require quadratically many parameters. Instead of explicit lower-order terms, we propose *augmenting* $\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}$, and $\bar{\boldsymbol{c}}$ with a constant feature of 1, which has approximately the same effect, while benefiting from the parameter sharing in the low-rank factorization; an effect described by Blondel et al. (2016). Siblings and co-parents factors are similarly parametrized with their own tensors.

7.6 Experiments

We implement structured output prediction models using both **linear** and **LSTM-based** parametrizations. We compare our proposed models to equivalent independent unary classifiers. For the RNN unstructured baseline, we compute unary potentials in the same way as in the structured model, but apply independent hinge losses at each variable, instead of the global structured hinge loss. Since the RNN weights are shared, this is a form of multi-task learning.

The overall trend is that training using a structured objective outperforms unstructured baseline models, even when structured inference is applied on the baseline predictions. While feature engineering generally outperforms neural models, we find that RNNs shine on proposition classification, and that structured training helps in making them competitive for argumentation mining.

We evaluate our proposed models on both datasets. For model selection and development we used k-fold cross-validation at document level: on CDCP we set $k = 3$ to avoid small validation folds, while on UKP we follow Stab and Gurevych (2017) setting $k = 5$. We compare our proposed structured learning systems (the linear structured SVM and the structured RNN) to the corresponding baseline versions.

We organize our experiments in three incremental variants of our factor graph: *basic*, *full*, and *strict*, each with the following components:[8]

component	basic	full	strict	(baseline)
unaries	✓	✓	✓	✓
compat. factors	✓	✓	✓	
compat. features		✓	✓	
higher-order		✓	✓	
link structure		✓	✓	✓
strict constraints			✓	✓

Following Stab and Gurevych (2017), we compute F_1 scores at proposition and link level, and also report their average as a summary of overall performance.[9] The

[8] Components are described in Section 7.4. The baselines *with inference* support only unaries and factors with no parameters, as indicated in the last column.

[9] For link F_1 scores, however, we find it more intuitive to only consider retrieval of positive links rather than macro-averaged two-class scores.

	Baseline						Structured					
	SVM			RNN			SVM			RNN		
Metric	basic	full	strict	basic	full	strict	basic	full	strict	basic	full	strict
CDCP dataset												
Average	47.4	47.3	47.9	40.8	38.0	38.0	48.1	49.3	**50.0**	43.5	33.5	38.2
Link (272)	22.0	21.9	23.8	9.9	12.8	12.8	24.7	25.1	**26.7**	14.4	14.6	10.5
Proposition	72.7	72.7	72.0	71.8	63.2	63.2	71.6	**73.5**	73.2	72.7	52.4	65.9
VALUE (491)	75.3	75.3	74.4	74.1	74.8	74.8	73.4	75.7	76.4	73.7	73.1	69.7
POLICY (153)	78.7	78.7	78.5	74.3	72.2	72.2	72.3	77.3	76.8	73.9	74.4	76.8
TESTIMONY (204)	70.3	70.3	68.6	74.6	71.8	71.8	69.8	71.7	71.5	74.2	72.3	75.8
FACT (124)	39.2	39.2	38.3	35.8	30.5	30.5	42.4	42.5	41.3	41.5	42.2	40.5
REFERENCE (1)	100.0	100.0	100.0	100.0	66.7	66.7	100.0	100.0	100.0	100.0	0.0	66.7
UKP dataset												
Average	64.7	66.6	66.5	58.7	57.4	58.7	67.1	**68.9**	67.1	59.0	63.6	64.7
Link (809)	55.8	59.7	**60.3**	44.8	43.8	44.0	56.9	60.1	56.9	44.1	50.4	50.1
Proposition	73.5	73.5	72.6	72.6	70.9	73.3	77.2	77.6	77.3	74.0	76.9	**79.3**
MAJOR CLAIM (153)	76.7	76.7	77.6	81.4	75.1	81.3	77.0	78.2	80.0	83.6	84.6	88.3
CLAIM (304)	55.4	55.4	52.0	51.7	52.7	53.5	64.3	64.5	62.8	53.2	60.2	62.0
PREMISE (809)	88.4	88.4	88.3	84.8	84.8	85.2	90.3	90.2	89.2	85.0	85.9	87.6

Table 7.2: Test set F_1 scores for link and proposition classification, as well as their averages. The number of test instances is shown in parentheses; best scores on overall tasks are in bold. Scores are micro-averaged at the document level.

results of a single prediction run on the test set are displayed in Table 7.2. The overall trend is that training using a structured objective is better than the baseline models, even when structured inference is applied on the baseline predictions. On UKP, for link prediction, the linear baseline can reach good performance when using inference, similar to the approach of Stab and Gurevych (2017), but the improvement in proposition prediction leads to higher overall F_1 for the structured models. Meanwhile, on the more difficult CDCP setting, performing inference on the baseline output is not competitive. While feature engineering still outperforms our RNN model, we find that RNNs shine on proposition classification, especially on UKP, and that structured training can make them competitive, reducing their observed lag on link prediction (Katiyar and Cardie, 2016), possibly through mitigating class imbalance.

7.7 Error Analysis

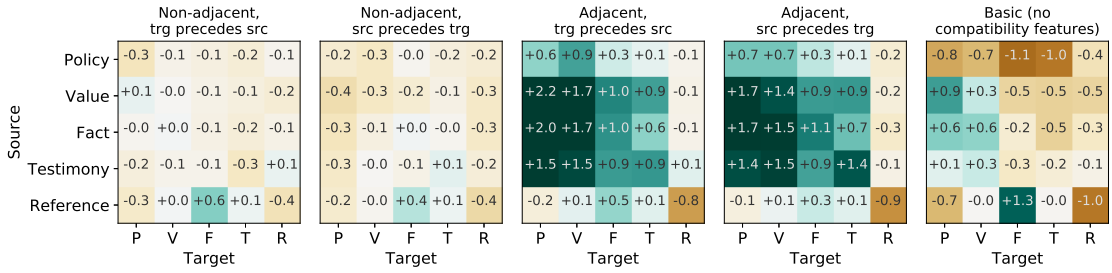


Figure 7.3: Learned conditional log-odds $\log \frac{p(\texttt{on}|\cdot)}{p(\texttt{off}|\cdot)}$, given the source and target proposition types and compatibility feature settings. First four figures correspond to the four possible settings of the compatibility features in the full structured SVM model. For comparison, the rightmost figure shows the same parameters in the basic structured SVM model, which does not use compatibility features.

Contribution of compatibility features. The compatibility factor in our model can be visualized as conditional odds ratios given the source and target

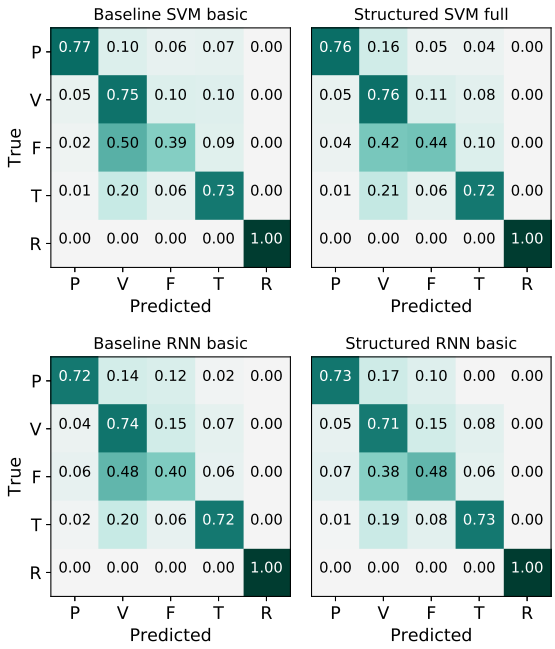


Figure 7.4: Normalized confusion matrices for proposition type classification.

proposition types. Since there are only four possible configurations of the compatibility features, we can plot all cases in Figure 7.3, alongside the basic model. Not using compatibility features, the basic model can only learn whether certain configurations are more likely than others (e.g. a REFERENCE supporting another REFERENCE is unlikely, while a REFERENCE supporting a FACT is more likely; essentially a soft version of our domain-specific strict constraints. The full model with compatibility features is finer grained, capturing, for example, that links from REFERENCE to FACT are more likely when the reference comes *after*, or that links from VALUE to POLICY are extremely likely only when the two are adjacent.

Proposition errors. The confusion matrices in Figure 7.4 reveal that the most common confusion is misclassifying FACT as VALUE. The strongest difference between the various models tested is that the RNN-based models make this error less often. For instance, in the proposition:

> And the single most frequently used excuse of any debtor is "I didn't receive the letter/invoice/statement"

the pronouns in the nested quote may be mistaken for subjectivity, leading to the structured SVMs predictions of VALUE or TESTIMONY, while the basic structured RNN correctly classifies it as FACT.

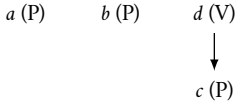
Link errors. While structured inference certainly helps baselines by preventing invalid structures such as cycles, it still depends on local decisions, losing to fully structured training in cases where joint proposition and link decisions are needed. For instance, in the following conclusion of an UKP essay, the annotators found no links:

> In short, [the individual should finance his or her education]$_a$ because [it is a personal choice.]$_b$ Otherwise, [it would cause too much cost from taxpayers and the government.]$_c$

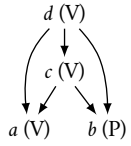
Indeed, no reasons are provided, but baseline are misled by the connectives: the SVM baseline outputs that b and c are PREMISES supporting the CLAIM a. The full structured SVM combines the two tasks and correctly recognizes the link structure.

Linear SVMs are still a very good baseline, but they tend to overgenerate links due to class imbalance, even if we use class weights during training. Surprisingly, RNNs are at the opposite end, being extremely conservative, and getting the highest precision among the models. On CDCP, where the number of true links is 272, the linear baseline with strict inference predicts 796 links with a precision of only 16%, while the strict structured RNN only predicts 52 links, with 33% precision; the example in Figure 7.5 illustrates this. In terms of higher-order structures, we find

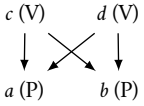
[I think the cost of education needs to be reduced (...) or repayment plans need to be income based.]$_a$ [As far as consumer protection, legal aid needs to be made available, affordable and effective,]$_b$ [and consumers need to take time to really know their rights and stop complaining about harassment]$_c$ [because that's a completely different cause of action than restitution.]$_d$



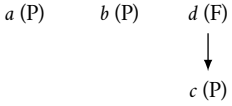
(a) Ground truth



(b) Baseline linear strict



(c) Structured linear full



(d) Structured RNN strict

Figure 7.5: Predictions on a CDCP comment where the structured RNN outperforms the other models.

that using higher-order factors increases precision, at a cost in recall. This is most beneficial for the 856 co-parent structures in the UKP test set: the full structured SVM has 53% F_1, while the basic structured SVM and the basic baseline get 47% and 45% respectively. On CDCP, while higher-order factors help, performance on siblings and co-parents is below 10% F_1 score. This is likely due to link sparsity and suggests plenty of room for further development.

Finally, we note that the strict constraints seem to help more with longer documents: over the comments with more than ten propositions, the link F_1 of the full linear model drops by 5% while the strict version only drops by 2.5%

7.8 Chapter Summary

We introduce an argumentation parsing model based on AD^3 relaxed inference in expressive factor graphs, experimenting with both linear structured SVMs and structured RNNs, parametrized with higher-order factors and link structure constraints. We demonstrate our model on a new argumentation mining dataset with more permissive argument structure annotation. Our model also achieves state-of-the-art link prediction performance on the UKP essays dataset. While we focus on monological argumentation, our model could be extended to dialogues, for which argumentation theory thoroughly motivates non-tree structures (Afantenos and Asher, 2014).

CHAPTER 8

CONCLUSIONS

To wrap up, in this chapter, we review the results and techniques that we developed, and discuss their consequences, impact, and directions for future work.

8.1 Summary and Discussion

Over the course of this work, we took an in-depth look at the possible uses of structure in machine learning models for NLP. Our experimental results consistently confirm that structure plays a crucial role in natural language tasks, as structure-aware models consistently outperform unstructured variants across many tasks.

The significance of our technical contributions lies across two fronts. On one side, many of our constructions are generalizations or extensions of significant pieces of the machine learning toolkit. Our sparse and structured attention mechanisms (Chapter 3) form a general family which includes softmax and sparsemax, and our Fenchel-Young losses provide a similarly extensible family of loss functions, which includes well-known losses such as the logistic, hinge, CRF, and structured SVM losses. Generalizations of this form are useful for shedding new light and new context onto existing concepts, leading to practical new tools as well as to new insights: for instance, generalizing softmax and sparsemax allowed us to rigorously capture why the former is always dense and the latter tends to be sparse.

On the other front, our work pushes the boundary of what transformations can be used as neural hidden layers, when training with backpropagation. By using *smoothing*, we develop differentiable transformations based on discrete and sparsity-inducing optimization problems, standing in contrast with the typical

deep learning tradition of purely continuous and dense transformations.

A common theme in our work is sparsity, which allows us to develop tractable and interpretable methods for latent structure, be it through structured sparsity (Chapter 3) or through graphical model inference (Chapter 4). We thus follow in the footsteps of recent work exploring more and more the use of structure in neural network outputs (Collobert et al., 2011; Kong et al., 2016) and in attention mechanisms (Kim et al., 2017; Liu and Lapata, 2018). By making use of sparsity, we are able empower machine learning models with new levels of performance, expressiveness, ease of use, and interpretability.

8.2 Future Directions

Approximate differentiable inference in overlapping factor graphs. We have shown that `SparseMAP` provides efficient, sparse, and differentiable inference in any structured model, as long as we have access to a MAP oracle. As seen in Chapter 7, there are situations in which domain-specific structure can only be expressed through overlapping factors, necessarily making exact MAP and marginal inference intractable. It remains to be seen what improvements an approximate `SparseMAP` loss can lead to for structured output models. Deriving an efficient backward pass for latent structure with overlapping factors constitutes a challenge.

User-friendly methods for defining structured sparsity. For combinatorial structure, `SparseMAP` provides an efficient forward and backward pass defined only in terms of a user-specified MAP inference procedure. Such a user-friendly way of defining new structured sparsity mappings is still elusive: proposing new

transformations like fusedmax and oscarmax requires case-by-case study. Recent developments in atomic norms for structured sparsity (Chandrasekaran et al., 2012) may lead to useful generalized formalisms, allowing for similar user-friendliness as we are able to achieve for `SparseMAP`.

Properties of learning with `SparseMAP` losses. The empirical results in Chapter 6 suggest good convergence and margin-like generalization properties. In the unstructured setting, the sparsemax loss (Martins and Astudillo, 2016) enjoys a margin property. As such, further study of the structured setting of Fenchel-Young losses, as well as the study of specialized learning algorithms for such losses, is an interesting direction for future research.

Efficient algorithms for structured sparsity. The projection onto the probability simplex shows up as a fundamental building block throughout our methods for latent structured sparsity in Chapter 3. For fast neural network prediction and training, developing an efficient GPU-parallelized algorithm for projecting onto the simplex, as well as for similar projections and optimization problems, has the potential to lead to important speedups.

Gradient-free learning of latent structure. In this work, we have focused on the most common way of training neural networks: stochastic gradient learning. An alternative that has not been explored very much in the literature is based on lagrangian dynamics (Carreira-Perpiñán and Wang, 2014; Taylor et al., 2016), replacing gradient computation with other types of subproblems. It remains to be seen whether such methods are competitive for latent structure applications.

APPENDIX A

PROOFS

A.1 Sparsity of attention mappings: Proof of Proposition 3.1

Recall that, by definition, $\Pi_\Omega(\theta) = \boldsymbol{p}$ iff. $\boldsymbol{p} \in \arg\min_{\boldsymbol{p}' \in \triangle^d} \Omega(\boldsymbol{p}') - \langle \theta, \boldsymbol{p}' \rangle$. The Lagrangian associated with this minimization problem is

$$\mathcal{L}(\boldsymbol{p}, \mu, \tau) = \Omega(\boldsymbol{p}) - \langle \theta + \mu, \boldsymbol{p} \rangle + \tau(\mathbf{1}^\top \boldsymbol{p} - 1). \tag{A.1}$$

Since $\operatorname{relint} \triangle^d \neq \varnothing$ (Equation 2.5), Slater's condition holds (Slater, 1959), thus the KKT conditions below are necessary and sufficient.

$$\begin{cases} 0 \in \partial_{\boldsymbol{p}} \mathcal{L}(\boldsymbol{p}, \mu, \tau) = \partial\Omega(\boldsymbol{p}) - \theta - \mu + \tau \mathbf{1}, \\ \langle \boldsymbol{p}, \mu \rangle = 0, \\ \boldsymbol{p} \in \triangle^d, \ \mu \geq 0, \end{cases} \tag{A.2}$$

where the addition and subtraction in the first condition are extended to sets.

For the forward implication, for a given $\boldsymbol{p} \in \triangle^d$, we seek θ such that $(\boldsymbol{p}, \mu, \tau)$ are a solution to the KKT conditions for some $\mu \geq 0$ and $\tau \in \mathbb{R}$. We will show that such θ exists by simply choosing $\mu = \mathbf{0}$ and $\tau = 0$. Those choices are dual feasible and guarantee that the slackness complementary condition is satisfied. In this case, we have from the first condition that $\theta \in \partial\Omega(\boldsymbol{p})$. Since $\partial\Omega(\boldsymbol{p})$ is non-empty for any $\boldsymbol{p} \in \triangle^d$, we can always find $\theta \in \mathbb{R}^d$ such that $(\boldsymbol{p}, \theta)$ are a dual pair, i.e., $\boldsymbol{p} = \nabla\Omega^*(\theta)$, which proves that $\nabla\Omega^*(\mathbb{R}^d) = \triangle^d$. Moreover, we can build infintely many such θ for other dual variable choices, showing that sparse solutions are not unlikely.

For the reverse implication, we must show that the subdifferential is nonempty. By assumption, for any $\boldsymbol{p} \in \triangle^d$ there exists $\theta \in \mathbb{R}^d$ such that $\Pi_\Omega(\theta) = \boldsymbol{p}$. Therefore,

there must exist μ and τ such that $(\boldsymbol{p}, \mu, \tau)$ satisfy the KKT conditions. From the first condition, $\theta + \mu - \tau\mathbf{1} \in \partial\Omega(\boldsymbol{p})$, completing the proof.

A.2 Jacobian for general differentiable Ω: Proof of Proposition 3.2

Recall that

$$\Pi_\Omega(\theta) = \operatorname*{arg\,min}_{\boldsymbol{p} \in \triangle^d} f(\boldsymbol{p}), \quad \text{where} \quad f(\boldsymbol{p}) \coloneqq \gamma\Omega(\boldsymbol{p}) - \boldsymbol{p}^\top\theta. \tag{A.3}$$

The optimum satisfies the fixed point iteration (Parikh and Boyd, 2014, Section 4.2)

$$\boldsymbol{p}^\star = \mathbf{P}_{\triangle^d}(\boldsymbol{p}^\star - \nabla f(\boldsymbol{p}^\star)). \tag{A.4}$$

Seeing $\boldsymbol{p}^\star$ as a function of θ, and $\mathbf{P}_{\triangle^d}$ and ∇f as functions of their inputs, we can apply the chain rule to (A.4) to obtain

$$\boldsymbol{J}_{\Pi_\Omega}(\theta) = \boldsymbol{J}_{\mathbf{P}_{\triangle^d}}\left(\boldsymbol{p}^\star - \nabla f(\boldsymbol{p}^\star)\right)\left(\boldsymbol{J}_{\Pi_\Omega}(\theta) - \boldsymbol{J}_{\nabla f \circ \boldsymbol{p}^\star}(\theta)\right). \tag{A.5}$$

Applying the chain rule once again to $\nabla f(\boldsymbol{p}^\star) = \gamma\nabla\Omega(\boldsymbol{p}^\star) - \theta$, we obtain

$$\begin{aligned} \boldsymbol{J}_{\nabla f \circ \boldsymbol{p}^\star}(\theta) &= \gamma \boldsymbol{J}_{\nabla\Omega}(\boldsymbol{p}^\star)\boldsymbol{J}_{\Pi_\Omega}(\theta) - \boldsymbol{I} \\ &= \gamma \boldsymbol{H}_\Omega(\boldsymbol{p}^\star)\boldsymbol{J}_{\Pi_\Omega}(\theta) - \boldsymbol{I}. \end{aligned} \tag{A.6}$$

Plugging this into (A.5) and re-arranging, we obtain

$$\left(\boldsymbol{I} + \boldsymbol{A}(\boldsymbol{B} - \boldsymbol{I})\right)\boldsymbol{J}_{\Pi_\Omega}(\theta) = \boldsymbol{A}, \tag{A.7}$$

where we defined the shorthands

$$\begin{aligned} \boldsymbol{A} &\coloneqq \boldsymbol{J}_{\mathbf{P}_{\triangle^d}}(\boldsymbol{p}^\star - \gamma\nabla\Omega(\boldsymbol{p}^\star) + \theta) \\ \boldsymbol{B} &\coloneqq \gamma \boldsymbol{H}_\Omega(\boldsymbol{p}^\star). \end{aligned} \tag{A.8}$$

A.3 Fusedmax and Oscarmax Jacobian: Proof of Proposition 3.4

Proof outline. Let $\boldsymbol{z}^\star = \mathbf{prox}_{\mathrm{TV}}(\boldsymbol{\theta})$ or $\mathbf{prox}_{\mathrm{OSC}}(\boldsymbol{\theta})$. We use the optimality conditions of $\mathbf{prox}_{\mathrm{TV}}$, respectively $\mathbf{prox}_{\mathrm{OSC}}$ in order to express $\boldsymbol{z}^\star$ as an explicit function of $\boldsymbol{\theta}$. Then, obtaining the Jacobians of $\mathbf{prox}_{\mathrm{TV}}(\boldsymbol{\theta})$ and $\mathbf{prox}_{\mathrm{OSC}}(\boldsymbol{\theta})$ follows by application of the chain rule to the two expressions. We discuss the proof for points where $\mathbf{prox}_{\mathrm{TV}}$ and $\mathbf{prox}_{\mathrm{OSC}}$ are differentiable; on the (zero-measure) set of nondifferentiable points (i.e. where the group structure changes) we may take one of Clarke's generalized gradients (Clarke, 1990).

Jacobian of $\mathbf{prox}_{\mathrm{TV}}$.

Lemma A.1 *Let $\boldsymbol{z}^\star = \mathbf{prox}_{TV}(\boldsymbol{\theta}) \in \mathbb{R}^d$ and $G_i^\star$ be the set of indices around i with the same value at the optimum, as defined in Section 3.5. Then, we have*

$$z_i^\star = \frac{\sum_{j \in G_i^\star} \theta_j + \lambda(s_{a_i} - s_{b_i})}{|G_i^\star|}, \tag{A.9}$$

where $a_i = \min G_i^\star, b_i = \max G_i^\star$ are the boundaries of segment $G_i^\star$, and

$$s_{a_i} = \begin{cases} 0 & \text{if } a = 1, \\ \mathsf{sign}(z^\star_{a_i - 1} - z_i^\star) & \text{if } a > 1 \end{cases} \quad \text{and} \quad s_{b_i} = \begin{cases} 0 & \text{if } b = d, \\ \mathsf{sign}(z_i^\star - z^\star_{b_i + 1}) & \text{if } b < d \end{cases}. \tag{A.10}$$

To prove Lemma A.1, we make use of the optimality conditions of the fused lasso proximity operator (Friedman et al., 2007, Equation 27), which state that $\boldsymbol{z}^\star$ satisfies

$$z_j^\star - \theta_j + \lambda(t_j - t_{j+1}) = 0 \tag{A.11}$$

where we denote

$$t_j \in \begin{cases} \{0\} & \text{if } i \in \{1, d\}, \\ \{\mathsf{sign}(z_j^\star - z_{j-1}^\star)\} & \text{if } z_j^\star \neq z_{j-1}^\star, \\ [-1, 1] & \text{o.w.} \end{cases} \qquad \forall j \in [d].$$

The optimality conditions (A.11) form a system with unknowns $z_j^\star, t_j$ for $j \in [d]$. To express $\boldsymbol{z}^\star$ as a function of $\boldsymbol{\theta}$, we shall now proceed to eliminate the unknowns t_j.

Let us focus on a particular segment $G_i^\star$. For readability, we drop the segment index i and use the shorthands $z \coloneqq z_i^\star, a \coloneqq a_i$, and $b \coloneqq b_i$. By definition, a and b satisfy

$$z_j^\star = z \quad \forall a \leq j \leq b, \qquad z_{a-1}^\star \neq z \text{ if } a > 1, \qquad z_{b+1}^\star \neq z \text{ if } b < d. \tag{A.12}$$

It immediately follows from the definition of t_j in (A.11) that

$$t_a = \begin{cases} 0 & \text{if } a = 1, \\ \mathsf{sign}(z - z_{a-1}^\star) & \text{if } a > 1 \end{cases} \quad \text{and} \quad t_{b+1} = \begin{cases} 0 & \text{if } b = d, \\ \mathsf{sign}(z_{b+1}^\star - z) & \text{if } b < d \end{cases}. \tag{A.13}$$

In other words, the unknowns t_a and t_b are already uniquely determined. To emphasize that they are known, we introduce $s_a \coloneqq t_a$ and $s_b \coloneqq t_{b+1}$, leaving t_j only unknown for $a < j \leq b$.

By rearranging the optimality conditions (A.11) we obtain the recursion

$$\lambda t_j = \theta_j - z + \lambda t_{j+1} \quad \forall a \leq j \leq b. \tag{A.14}$$

We start with the first equation in the segment (at $j = a$), and unroll the recursion

until reaching the stopping condition $j = b$.

$$\begin{aligned} \lambda s_a &= \theta_a - z + \lambda t_{a+1} \\ &= \theta_a - z + \theta_{a+1} - z + \cdots + \theta_b - z + \lambda s_b \\ &= \sum_{k=a}^{b} \theta_k - (b - a + 1)z + \lambda s_b \end{aligned} \tag{A.15}$$

Rearranging the terms, we obtain the expression

$$z = \frac{\sum_{k=a}^{b} \theta_k + \lambda(s_b - s_a)}{b - a + 1}. \tag{A.16}$$

Applying this calculation to each segment in $\boldsymbol{z}^\star$ yields the desired result. □

The proof of Proposition 3.4 follows by applying the chain rule to (A.9), noting that the groups G_i^* are constant within a neighborhood of $\boldsymbol{\theta}$ (observation also used for OSCAR in (Bondell and Reich, 2008)). Therefore, for $\mathbf{prox}_{\mathrm{TV}}$,

$$\frac{\partial z_i^\star}{\partial \theta_j} = \frac{1}{|G_i^\star|} \left(\sum_{k \in G_i^\star} \frac{\partial \theta_k}{\partial \theta_j} + \lambda \left(\frac{\partial s_b}{\partial \theta_j} - \frac{\partial s_a}{\partial \theta_j} \right) \right). \tag{A.17}$$

Since s_b and s_a are either constant or sign functions w.r.t. $\boldsymbol{\theta}$, their partial derivatives are 0, and thus

$$\frac{\partial z_i^\star}{\partial \theta_j} = \begin{cases} \frac{1}{|G_i^\star|} & \text{if } j \in G_i^\star, \\ 0 & \text{o.w.} \end{cases}. \tag{A.18}$$

Jacobian of $\mathbf{prox}_{\mathrm{OSC}}$.

Lemma A.2 *((Zeng and Figueiredo, 2014, Theorem 1), (Zhong and Kwok, 2012, Proposition 3)) Let $\boldsymbol{z}^\star = \mathbf{prox}_{OSC}(\boldsymbol{\theta}) \in \mathbb{R}^d$ and $G_i^\star$ be the set of indices around i with the same value at the optimum: $G_i^\star = \{j \in [d] : |z_i^\star| = |z_j^\star|\}$. Then, we have*

$$z_i^\star = \mathsf{sign}(\theta_i) \max \left(\frac{\sum_{j \in G_i^\star} |\theta_j|}{|G_i^\star|} - w_i, 0 \right), \tag{A.19}$$

$$\text{where } w_i = \lambda\left(d - \frac{u_i + v_i}{2}\right), \quad u_i = \left|\{j \in [d] : |z_j^\star| < |z_i^\star|\}\right|, \quad v_i = u_i + |G_i^\star|. \tag{A.20}$$

Lemma A.2 is a simple reformulation of Theorem 1, part *ii* from (Zeng and Figueiredo, 2014). With the same observation that the induced groups do not change within a neighborhood of $\boldsymbol{\theta}$, we may differentiate (A.19) to obtain

$$\frac{\partial z_i^\star}{\partial \theta_j} = \begin{cases} 0 & \text{if } z_i^\star = 0, \\ \dfrac{\mathsf{sign}(\theta_i)}{|G_i^\star|} \displaystyle\sum_{k \in G_i^\star} \frac{\partial |\theta_k|}{\partial \theta_k} \frac{\partial \theta_k}{\partial \theta_j} - \frac{\partial w_i}{\partial \theta_j} & \text{o.w.} \end{cases}. \tag{A.21}$$

Noting that $\frac{\partial w_i}{\partial \theta_j} = 0$, as w_i is derived only from group indices and the term $\frac{\partial |\theta_k|}{\partial \theta_k} \frac{\partial \theta_k}{\partial \theta_j}$ either vanishes (when $k \neq j$) or else equals $\mathsf{sign}(x_j)$ with $\theta_j \neq 0$, we substitute $\mathsf{sign}(z_j^\star)$ for $\mathsf{sign}(\theta_j)$ (Zeng and Figueiredo, 2014) to get

$$\frac{\partial z_i^\star}{\partial \theta_j} = \begin{cases} \dfrac{\mathsf{sign}(z_i^\star z_j^\star)}{|G_i^\star|} & \text{if } j \in G_i^\star \text{ and } z_i^\star \neq 0, \\ 0 & \text{o.w.} \end{cases}. \tag{A.22}$$

A.4 Computing the SparseMAP Jacobian: Proof of Proposition 4.1

Recall that SparseMAP is defined as the $\boldsymbol{u}^\star$ that maximizes the value of the quadratic program (Equation 4.7),

$$g(\boldsymbol{\eta}_u, \boldsymbol{\eta}_v) \coloneqq \max_{[\boldsymbol{u};\boldsymbol{v}] \in \mathcal{M}_A} \boldsymbol{\eta}_u{}^\top \boldsymbol{u} + \boldsymbol{\eta}_v{}^\top \boldsymbol{v} - \frac{1}{2}\|\boldsymbol{u}\|_2^2. \tag{A.23}$$

As the ℓ_2^2 norm is strongly convex, there is always a unique minimizer $\boldsymbol{u}^\star$ (implying that SparseMAP is well-defined), and the convex conjugate of the QP in (A.23), $g^*(\boldsymbol{u}, \boldsymbol{v}) = \left\{\frac{1}{2}\|\boldsymbol{u}\|_2^2, [\boldsymbol{u};\boldsymbol{v}] \in \mathcal{M}_A; -\infty \text{ otherwise}\right\}$ is smooth in $\boldsymbol{u}$, implying that

SparseMAP (which only returns $\boldsymbol{u}$) is Lipschitz-continuous and thus differentiable almost everywhere.

We now rewrite the QP in Equation A.23 in terms of the convex combination of vertices of the marginal polytope

$$\min_{\boldsymbol{p} \in \triangle^D} \frac{1}{2} \|\boldsymbol{M}\boldsymbol{p}\|_2^2 - \boldsymbol{\theta}^\top \boldsymbol{p} \qquad \text{where } \boldsymbol{\theta} \coloneqq \boldsymbol{A}^\top \boldsymbol{\eta} \tag{A.24}$$

We use the optimality conditions of problem A.24 to derive an explicit relationship between $\boldsymbol{u}^\star$ and $\boldsymbol{x}$. At an optimum, the following KKT conditions hold

$$\boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{p}^\star - \boldsymbol{\lambda}^\star + \tau^\star \mathbf{1} = \boldsymbol{\theta} \tag{A.25}$$

$$\mathbf{1}^\top \boldsymbol{p}^\star = 1 \tag{A.26}$$

$$\boldsymbol{p}^\star \geq \mathbf{0} \tag{A.27}$$

$$\boldsymbol{\lambda}^\star \geq \mathbf{0} \tag{A.28}$$

$$\boldsymbol{\lambda}^{\star\top} \boldsymbol{p}^\star = 0 \tag{A.29}$$

Let $\bar{\mathcal{Y}}$ denote the support of $\boldsymbol{p}^\star$, *i. e.*, $\bar{\mathcal{Y}} = \{y \ : \ p_y^\star > 0\}$. From Equation A.29 we have $\boldsymbol{\lambda}_{\bar{\mathcal{Y}}} = \mathbf{0}$ and therefore

$$\boldsymbol{M}_{\bar{\mathcal{Y}}}{}^\top \boldsymbol{M}_{\bar{\mathcal{Y}}} \boldsymbol{p}_{\bar{\mathcal{Y}}}^\star + \tau^\star \mathbf{1} = \boldsymbol{\theta}_{\bar{\mathcal{Y}}} \tag{A.30}$$

$$\mathbf{1}^\top \boldsymbol{p}_{\bar{\mathcal{Y}}}^\star = 1 \tag{A.31}$$

Solving for $\boldsymbol{p}_{\bar{\mathcal{Y}}}^\star$ in Equation A.30 we get a direct expression

$$\boldsymbol{p}_{\bar{\mathcal{Y}}}{}^\star = (\boldsymbol{M}_{\bar{\mathcal{Y}}}{}^\top \boldsymbol{M}_{\bar{\mathcal{Y}}})^{-1}(\boldsymbol{\theta}_{\bar{\mathcal{Y}}} - \tau^\star \mathbf{1}) = \boldsymbol{Z}(\boldsymbol{\theta}_{\bar{\mathcal{Y}}} - \tau^\star \mathbf{1}).$$

where we introduced $\boldsymbol{Z} = (\boldsymbol{M}_{\bar{\mathcal{Y}}}^\top \boldsymbol{M}_{\bar{\mathcal{Y}}})^{-1}$. Solving for $\tau^\star$ yields

$$\tau^\star = \frac{1}{\mathbf{1}^T \boldsymbol{Z} \mathbf{1}} \left(\mathbf{1}^T \boldsymbol{Z} \boldsymbol{\theta}_{\bar{\mathcal{Y}}} - 1 \right)$$

Plugging this back and left-multiplying by $\boldsymbol{M}_{\bar{\mathcal{Y}}}$ we get

$$\boldsymbol{u}^\star = \boldsymbol{M}_{\bar{\mathcal{Y}}} \boldsymbol{p}_{\bar{\mathcal{Y}}} = \boldsymbol{M}_{\bar{\mathcal{Y}}} \boldsymbol{Z} \left(\boldsymbol{\theta}_{\bar{\mathcal{Y}}} - \frac{1}{\mathbf{1}^\top \boldsymbol{Z} \mathbf{1}} \mathbf{1}^\top \boldsymbol{Z} \boldsymbol{\theta}_{\bar{\mathcal{Y}}} \mathbf{1} + \frac{1}{\mathbf{1}^\top \boldsymbol{Z} \mathbf{1}} \mathbf{1} \right)$$

Note that, in a neighborhood of $\boldsymbol{\eta}$, the support of the solution $\bar{\mathcal{Y}}$ is constant. (On the measure-zero set of points where the support changes, `SparseMAP` is subdifferentiable and our assumption yields a generalized Jacobian (Clarke, 1990).) Differentiating w.r.t. the score of a configuration θ_y, we get the expression

$$\frac{\partial \boldsymbol{u}^\star}{\partial \theta_y} = \begin{cases} \boldsymbol{M} \left(\boldsymbol{I} - \frac{1}{\mathbf{1}^T \boldsymbol{Z} \mathbf{1}} \boldsymbol{Z} \mathbf{1} \mathbf{1}^T \right) \boldsymbol{z}_y & y \in \bar{\mathcal{Y}} \\ \mathbf{0} & y \notin \bar{\mathcal{Y}} \end{cases} \tag{A.32}$$

Since $\theta_y = \boldsymbol{a}_y^\top \boldsymbol{\eta}$, by the chain rule, we get the desired result

$$\frac{\partial \boldsymbol{u}^\star}{\partial \boldsymbol{\eta}} = \frac{\partial \boldsymbol{u}^\star}{\partial \boldsymbol{\theta}} \boldsymbol{A}^\top. \tag{A.33}$$

A.5 Distribution-wise SparseMAP Jacobian: Proof of Proposition 5.1

We follow the derivation from Appendix A.4, with the latent-variable notation from Chapter 5. Any solution $p_{\boldsymbol{w}}$ satisfies, for any $y \in \bar{\mathcal{Y}}(x)$

$$p_{\boldsymbol{w}}(y|x) = \sum_{y' \in \bar{\mathcal{Y}}(x)} z_{y,y'} (\theta_{\boldsymbol{w}}(y'; x) - \tau^\star), \tag{A.34}$$

where

$$\tau^\star = \frac{-1 + \sum_{\{y'', y'\} \in \bar{\mathcal{Y}}(x)} z_{y'',y'} \theta_{\boldsymbol{w}}(y'; x)}{\zeta}. \tag{A.35}$$

To simplify notation, we denote $p_{\boldsymbol{w}}(y|x) = p_1(\boldsymbol{w}) - p_2(\boldsymbol{w})$ where

$$\begin{aligned} p_1(\boldsymbol{w}) &\coloneqq \sum_{y' \in \bar{\mathcal{Y}}(x)} z_{y,y'} \; f_{\boldsymbol{w}}(y'; x), \\ p_2(\boldsymbol{w}) &\coloneqq \Big(\sum_{y'} z_{y,y'} \Big) \cdot \tau^\star \\ &= \varsigma(y) \cdot \zeta^{-1} \Big(\sum_{y'', y'} z_{y'',y'} f_{\boldsymbol{w}}(y'; x) \Big) - \text{const.} \end{aligned} \tag{A.36}$$

Differentiating, we get

$$\begin{aligned}\frac{\partial p_1}{\partial \boldsymbol{w}} &= \sum_{y' \in \bar{\mathcal{Y}}(x)} z_{y,y'} \frac{\partial \theta_{\boldsymbol{w}}(y'; x)}{\partial \boldsymbol{w}}, \\ \frac{\partial p_2}{\partial \boldsymbol{w}} &= \sum_{y' \in \bar{\mathcal{Y}}(x)} \varsigma(y) \cdot \zeta^{-1} \Big(\sum_{y''} z_{y'',y'} \Big) \frac{\partial \theta_{\boldsymbol{w}}(y'; x)}{\partial \boldsymbol{w}}. \\ &= \sum_{y' \in \bar{\mathcal{Y}}(x)} \varsigma(y) \cdot \zeta^{-1} \varsigma(y') \frac{\partial \theta_{\boldsymbol{w}}(y'; x)}{\partial \boldsymbol{w}}.\end{aligned} \tag{A.37}$$

Putting it all together, we obtain

$$\frac{\partial p_{\boldsymbol{w}}(y|x)}{\partial \boldsymbol{w}} = \sum_{y' \in \bar{\mathcal{Y}}(x)} \Big(z_{y,y'} - \zeta^{-1} \varsigma(y) \varsigma(y') \Big) \frac{\partial \theta_{\boldsymbol{w}}(y')}{\partial \boldsymbol{w}}, \tag{A.38}$$

which is the top branch of the conditional. For the other branch, observe that the support $\bar{\mathcal{Y}}(x)$ is constant within a neighborhood of $\boldsymbol{w}$, yielding $y \notin \bar{\mathcal{Y}}(x)$, $\frac{\partial p(y)}{\partial \boldsymbol{w}} = 0$. Importantly, since $\boldsymbol{Z}$ is computed as a side-effect of the SparseMAP forward pass, the backward pass computation is efficient.

A.6 Properties of Fenchel-Young Losses: Proof of Proposition 6.1

We recall that the structured Fenchel-Young loss defined by a convex $\Omega : \mathbb{R}^D \to \mathbb{R}$ and a matrix $\boldsymbol{A}$ is defined as

$$L_\Omega^{\boldsymbol{A}} : \mathbb{R}^d \times \triangle^D \to \mathbb{R}_+, \qquad L_\Omega^{\boldsymbol{A}}(\boldsymbol{\eta}, \bar{\boldsymbol{p}}) \coloneqq \Omega_\triangle^*(\boldsymbol{A}^\top \boldsymbol{\eta}) + \Omega_\triangle(\bar{\boldsymbol{p}}) - \boldsymbol{\eta}^\top \boldsymbol{A} \bar{\boldsymbol{p}}.$$

Since $\Omega_\triangle$ is the restriction of a convex function to a convex set, it is convex (Boyd and Vandenberghe, 2004, Section 3.1.2).

Property 1. From the Fenchel-Young inequality (Fenchel, 1949; Boyd and Vandenberghe, 2004, Section 3.3.2), we have

$$\boldsymbol{\theta}^\top \boldsymbol{p} \leq \Omega_\triangle^*(\boldsymbol{\theta}) + \Omega_\triangle(\boldsymbol{p}).$$

In particular, when $\eta = \boldsymbol{A}^\top \theta$,

$$\begin{aligned} 0 &\leq -\eta^\top \boldsymbol{A}\boldsymbol{p} + \Omega_\triangle^*(\boldsymbol{A}^\top \eta) + \Omega_\triangle(\boldsymbol{p}) \\ &= L_\Omega^{\boldsymbol{A}}(\eta, \boldsymbol{p}). \end{aligned}$$

Property 2. Equality is achieved when

$$\Omega_\triangle^*(\boldsymbol{A}^\top \eta) = \eta^\top \boldsymbol{A}\boldsymbol{p} - \Omega_\triangle(\boldsymbol{p}) \iff$$
$$\max_{\boldsymbol{p}' \in \triangle^D} \eta^\top \boldsymbol{A}\boldsymbol{p}' - \Omega(\boldsymbol{p}') = \eta^\top \boldsymbol{A}\boldsymbol{p} - \Omega(\boldsymbol{p}),$$

where we used the fact that $\boldsymbol{p} \in \triangle^D$. The claim immediately follows.

Property 3. To prove convexity in η, we rewrite the loss, for fixed $\boldsymbol{p}$, as

$$L_\Omega^{\boldsymbol{A}}(\eta) = h(\boldsymbol{A}^\top \eta) + \text{const}, \quad \text{where} \quad h(\theta) = \Omega_\triangle^*(\theta) - \theta^\top \boldsymbol{p}.$$

$\Omega_\triangle^\star$ is a convex conjugate, and thus itself convex. Linear functions are convex, and the sum of two convex functions is convex, therefore h is convex. Finally, the composition of a convex function with a linear function is convex as well, thus the function $\left(h\boldsymbol{A}^\top\right)$ is convex. Convexity of $L_\Omega^{\boldsymbol{A}}$ in η directly follows. Convexity in $\boldsymbol{y}$ is straightforward, as the sum of a convex and a linear function (Boyd and Vandenberghe, 2004, Sections 3.2.1, 3.2.2, 3.3.1).

Property 4. This follows from the scaling property of the convex conjugate (Boyd and Vandenberghe, 2004, Section 3.3.2)

$$(t\Omega)^*(\theta) = t\Omega^*(t^{-1}\theta)$$

Denoting $\boldsymbol{\eta}' = t^{-1}\boldsymbol{\eta}$, we have that

$$\begin{aligned} L^{\boldsymbol{A}}_{t\Omega}(\boldsymbol{\eta}, \boldsymbol{p}) &= (t\Omega_{\triangle})^*(\boldsymbol{A}^\top \boldsymbol{\eta}) + t\Omega_{\triangle}(\boldsymbol{p}) - \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} \\ &= t\Omega^*_{\triangle}(\boldsymbol{A}^\top \boldsymbol{\eta}') + t\Omega_{\triangle}(\boldsymbol{p}) - \boldsymbol{\eta}^\top \boldsymbol{A}\boldsymbol{p} \\ &= t\big(\Omega^*_{\triangle}(\boldsymbol{A}^\top \boldsymbol{\eta}') + \Omega_{\triangle}(\boldsymbol{p}) - \boldsymbol{\eta}'^\top \boldsymbol{A}\boldsymbol{p}\big) = tL^{\boldsymbol{A}}_{\Omega}(t^{-1}\boldsymbol{\eta}, \boldsymbol{p}). \end{aligned}$$

APPENDIX B

IMPLEMENTATION DETAILS

B.1 Conditional Gradient Algorithms for SparseMAP

We adapt the presentation of vanilla, away-step and pairwise conditional gradient of Lacoste-Julien and Jaggi (2015).

Recall the SparseMAP problem (Equation 4.7), which we rewrite below as a minimization, to align with the notation of Lacoste-Julien and Jaggi (2015)

$$\mathsf{SparseMAP}_A(\boldsymbol{\eta}) \coloneqq \operatorname*{arg\,min}_{\boldsymbol{u}:\, [\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}_A} f(\boldsymbol{u},\boldsymbol{v}), \qquad \text{where } f(\boldsymbol{u},\boldsymbol{v}) \coloneqq \frac{1}{2}\|\boldsymbol{u}\|_2^2 - {\boldsymbol{\eta}_u}^\top \boldsymbol{u} - {\boldsymbol{\eta}_v}^\top \boldsymbol{v}.$$

The gradients of the objective function f w.r.t. the two variables are

$$\nabla_{\boldsymbol{u}} f(\boldsymbol{u}', \boldsymbol{v}') = \boldsymbol{u}' - \boldsymbol{\eta}_u, \qquad \nabla_{\boldsymbol{v}} f(\boldsymbol{u}', \boldsymbol{v}') = -\boldsymbol{\eta}_v.$$

The ingredients required to apply conditional gradient algorithms are solving linear minimization problem, selecting the away step, computing the Wolfe gap, and performing line search.

Linear minimization problem. For SparseMAP, this amounts to a MAP inference call, since

$$\begin{aligned}
&\operatorname*{arg\,min}_{[\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}_A} \left\langle \nabla_{\boldsymbol{u}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{u} \right\rangle + \left\langle \nabla_{\boldsymbol{v}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{v} \right\rangle \\
&= \operatorname*{arg\,min}_{[\boldsymbol{u},\boldsymbol{v}]\in\mathcal{M}_A} (\boldsymbol{u}' - \boldsymbol{\eta}_u)^\top \boldsymbol{u} - {\boldsymbol{\eta}_v}^\top \boldsymbol{v} \\
&= \{[\boldsymbol{m}_y, \boldsymbol{n}_y] \;:\; y \in \mathsf{MAP}_A(\boldsymbol{\eta}_u - \boldsymbol{u}', \boldsymbol{\eta}_v)\}.
\end{aligned}$$

where we assume MAP_A yields the set of maximally-scoring structures.

Away step selection. This step involves searching the currently selected structures in the active set $\bar{\mathcal{Y}}$ with the *opposite* goal: finding the structure *maximizing* the linearization

$$\begin{aligned}
&\arg\max_{y \in \bar{\mathcal{Y}}} \ \left\langle \nabla_{\boldsymbol{u}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{m}_y \right\rangle + \left\langle \nabla_{\boldsymbol{v}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{n}_y \right\rangle \\
&= \arg\max_{y \in \bar{\mathcal{Y}}} \ (\boldsymbol{u}' - \boldsymbol{\eta}_u)^\top \boldsymbol{m}_y - \boldsymbol{\eta}_v{}^\top \boldsymbol{n}_y
\end{aligned}$$

Wolfe gap. The gap at a point $\boldsymbol{d} = [\boldsymbol{d}_{\boldsymbol{u}}, \boldsymbol{d}_{\boldsymbol{v}}]$ is given by

$$\begin{aligned}
\text{gap}(\boldsymbol{d}, \boldsymbol{u}') &:= \left\langle -\nabla_{\boldsymbol{u}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{d}_{\boldsymbol{u}} \right\rangle + \left\langle -\nabla_{\boldsymbol{v}} f(\boldsymbol{u}', \boldsymbol{v}'), \boldsymbol{d}_{\boldsymbol{v}} \right\rangle \\
&= \left\langle \boldsymbol{\eta}_u - \boldsymbol{u}', \boldsymbol{d}_{\boldsymbol{u}} \right\rangle + \left\langle \boldsymbol{\eta}_v, \boldsymbol{d}_{\boldsymbol{v}} \right\rangle.
\end{aligned} \tag{B.1}$$

Line search. Once we have picked a direction $\boldsymbol{d} = [\boldsymbol{d}_{\boldsymbol{u}}, \boldsymbol{d}_{\boldsymbol{v}}]$, we can pick the optimal step size by solving a simple optimization problem. Let $\boldsymbol{u}_\gamma := \boldsymbol{u}' + \gamma \boldsymbol{d}_{\boldsymbol{u}}$, and $\boldsymbol{v}_\gamma := \boldsymbol{v}' + \gamma \boldsymbol{d}_{\boldsymbol{v}}$. We seek γ so as to optimize

$$\arg\min_{\gamma \in [0, \gamma_{\max}]} f(\boldsymbol{u}_\gamma, \boldsymbol{v}_\gamma)$$

Setting the gradient w.r.t. γ to 0 yields

$$\begin{aligned}
0 &= \frac{\partial}{\partial \gamma} f(\boldsymbol{u}_\gamma, \boldsymbol{v}_\gamma) \\
&= \left\langle \boldsymbol{d}_{\boldsymbol{u}}, \nabla_{\boldsymbol{u}} f(\boldsymbol{u}_\gamma, \boldsymbol{v}_\gamma) \right\rangle + \left\langle \boldsymbol{d}_{\boldsymbol{v}}, \nabla_{\boldsymbol{v}} f(\boldsymbol{u}_\gamma, \boldsymbol{v}_\gamma) \right\rangle \\
&= \left\langle \boldsymbol{d}_{\boldsymbol{u}}, \boldsymbol{u}' + \gamma \boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\eta}_u \right\rangle + \left\langle \boldsymbol{d}_{\boldsymbol{v}}, -\boldsymbol{\eta}_v \right\rangle \\
&= \gamma \left\| \boldsymbol{d}_{\boldsymbol{u}} \right\|_2^2 + \boldsymbol{u}'^\top \boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\eta}^\top \boldsymbol{d}
\end{aligned}$$

We may therefore compute the optimal step size γ as

$$\gamma = \max\left(0, \min\left(\gamma_{\max}, \frac{\boldsymbol{\eta}^\top \boldsymbol{d} - \boldsymbol{u}'^\top \boldsymbol{d}_{\boldsymbol{u}}}{\left\| \boldsymbol{d}_{\boldsymbol{u}} \right\|_2^2}\right)\right) \tag{B.2}$$

Algorithm 3 Conditional gradient for SparseMAP

Initialize: $y^{(0)} \leftarrow \mathsf{MAP}_A(\boldsymbol{\eta}_u, \boldsymbol{\eta}_v)$; $\bar{\mathcal{Y}}^{(0)} = \{y^{(0)}\}$; $\boldsymbol{p}^{(0)} = \boldsymbol{e}_{y^{(0)}}$; $[\boldsymbol{u}^{(0)}, \boldsymbol{v}^{(0)}] = \boldsymbol{a}_{y^{(0)}}$
for $t = 0 \dots t_{\mathsf{max}}$ **do**
  $y \leftarrow \mathsf{MAP}_A(\boldsymbol{\eta}_u - \boldsymbol{u}^{(t)}, \boldsymbol{\eta}_v)$; $\quad \boldsymbol{d}^{\mathrm{F}} \leftarrow \boldsymbol{a}_y - [\boldsymbol{u}^{(t)}, \boldsymbol{v}^{(t)}]$ (forward)
  $w \leftarrow \arg\max_{w \in \bar{\mathcal{Y}}^{(t)}} (\boldsymbol{\eta}_u - \boldsymbol{u}^{(t)})^\top \boldsymbol{m}_w + \boldsymbol{\eta}_v{}^\top \boldsymbol{n}_w$; $\quad \boldsymbol{d}^{\mathrm{W}} \leftarrow [\boldsymbol{u}^{(t)}, \boldsymbol{v}^{(t)}] - \boldsymbol{a}_w$ (away)
  if $\mathrm{gap}(\boldsymbol{d}^{\mathrm{F}}, \boldsymbol{u}^{(t)}) < \epsilon$ **then**
    return $\boldsymbol{u}^{(t)}$ (Equation B.1)
  end if
  if variant = vanilla **then**
    $\boldsymbol{d} \leftarrow \boldsymbol{d}^{\mathrm{F}}$; $\quad \gamma_{\mathsf{max}} \leftarrow 1$
  else if variant = pairwise **then**
    $\boldsymbol{d} \leftarrow \boldsymbol{d}^{\mathrm{F}} + \boldsymbol{d}^{\mathrm{W}}$; $\quad \gamma_{\mathsf{max}} \leftarrow y_w$
  else if variant = away-step **then**
    if $\mathrm{gap}(\boldsymbol{d}^{\mathrm{F}}, \boldsymbol{u}^{(t)}) \geq \mathrm{gap}(\boldsymbol{d}^{\mathrm{W}}, \boldsymbol{u}^{(t)})$ **then**
      $\boldsymbol{d} \leftarrow \boldsymbol{d}^{\mathrm{F}}$; $\quad \gamma_{\mathsf{max}} \leftarrow 1$
    else
      $\boldsymbol{d} \leftarrow \boldsymbol{d}^{\mathrm{A}}$; $\quad \gamma_{\mathsf{max}} \leftarrow p_w/(1 - p_w)$
    end if
  end if
  Compute step size γ (Equation B.2)
  $[\boldsymbol{u}^{(t+1)}, \boldsymbol{v}^{(t+1)}] \leftarrow [\boldsymbol{u}^{(t)}, \boldsymbol{v}^{(t)}] + \boldsymbol{d}$
  Update $\bar{\mathcal{Y}}^{(t+1)}$ and $\boldsymbol{p}^{(t+1)}$ accordingly.
end for

BIBLIOGRAPHY

Ryan P Adams and Richard S Zemel. Ranking via Sinkhorn propagation. *arXiv e-prints*, 2011.

Stergos Afantenos and Nicholas Asher. Counter-argumentation and discourse: A case study. In *Proc. of ArgNLP*, 2014.

Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*, 2017.

Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3):145–373, 2013.

Francis Bach, Simon Lacoste-Julien, and Guillaume Obozinski. On the equivalence between herding and conditional gradient algorithms. In *Proc. of ICML*, 2012.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*, 2015.

Gökhan Bakır, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data*. The MIT Press, 2007.

Keith Ball, Eric A Carlen, and Elliott H Lieb. Sharp uniform convexity and smoothness inequalities for trace norms. *Inventiones Mathematicae*, 115(1):463–482, 1994.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-Sequence learning using gated graph neural networks. In *Proc. of ACL*, 2018.

David Belanger and Andrew McCallum. Structured prediction energy networks. In *Proc. of ICML*, 2016.

David Belanger, Dan Sheldon, and Andrew McCallum. Marginal inference in MRFs using Frank-Wolfe. In *Proc. of the NIPS Workshop on Greedy Optimization, Frank-Wolfe and Friends*, 2013.

David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *Proc. of ICML*, 2017.

Yoshua Bengio, Nicolas Le Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. In *Proc. of NIPS*, 2005.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. In *Proc. of NIPS*, 2013.

Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

Mathieu Blondel, Masakazu Ishihata, Akinori Fujino, and Naonori Ueda. Polynomial Networks and Factorization Machines: New Insights and Efficient Training Algorithms. In *Proc. of ICML*, 2016.

Mathieu Blondel, André FT Martins, and Vlad Niculae. Learning classifiers with Fenchel-Young losses: Generalized entropies, margins, and algorithms. *arXiv e-prints*, 2018.

Howard D Bondell and Brian J Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR. *Biometrics*, 64(1): 115–123, 2008.

Matko Bosnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable Forth interpreter. In *Proc. of ICML*, 2017.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proc. of EMNLP*, 2015.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proc. of ACL*, 2016.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Glenn W Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3, 1950.

Peter Brucker. An $O(n)$ algorithm for quadratic knapsack problems. *Operations Research Letters*, 3(3):163–166, 1984.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *Proc. of ICLR*, 2014.

Andreas Buja, Werner Stuetzle, and Yi Shen. Loss functions for binary class probability estimation and classification: Structure and applications. Technical report, University of Pennsylvania, 2005.

Paolo M Camerini, Luigi Fratta, and Francesco Maffioli. The k best spanning arborescences of a network. *Networks*, 10(2):91–109, 1980.

Miguel Á Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. In *Proc. of AISTATS*, 2014.

Venkat Chandrasekaran, Benjamin Recht, Pablo A Parrilo, and Alan S Willsky.

The convex geometry of linear inverse problems. *Foundations of Computational Mathematics*, 12(6):805–849, 2012.

Chandra R. Chegireddy and Horst W. Hamacher. Algorithms for finding K-best perfect matchings. *Discrete Applied Mathematics*, 18(2):155 – 165, 1987.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. In *Proc. of ACL*, 2017.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Proc. of NIPS*, 2015.

Yoeng-Jin Chu and Tseng-Hong Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

Frank H Clarke. *Optimization and Nonsmooth Analysis*. SIAM, 1990.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. Incorporating structural alignment biases into an attentional neural translation model. In *Proc. of NAACL-HLT*, 2016.

Michael Collins. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*, 2002.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

Laurent Condat. A direct algorithm for 1-d total variation denoising. *IEEE Signal Processing Letters*, 20(11):1054–1057, 2013.

Laurent Condat. Fast projection onto the simplex and the ℓ_1 ball. *Mathematical Programming*, 158(1-2):575–585, 2016.

Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering*, 15(4):i–xvii, 2009.

Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. In *Proc. of ICLR*, 2017.

John M Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

George B Dantzig, Alex Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

Dipanjan Das, Desai Chen, André FT Martins, Nathan Schneider, and Noah A Smith. Frame-semantic parsing. *Computational Linguistics*, 40(1):9–56, 2014.

A Philip Dawid. Probability forecasting. *Encyclopedia of Statistical Sciences*, 1986.

Morris H DeGroot. Uncertainty, information, and sequential experiments. *The Annals of Mathematical Statistics*, pages 404–419, 1962.

Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Proc. of NIPS*, 2017.

John C Duchi, Khashayar Khosravi, and Feng Ruan. Multiclass classification, information, divergence, and surrogate risk. *Annals of Statistics*, 2018.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Proc. of NIPS*, 2001.

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

Werner Fenchel. On conjugate convex functions. *Canad. J. Math*, 1(73-77), 1949.

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

Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007.

Menachem Fromer and Amir Globerson. An LP view of the M-best MAP problem. In *Proc. of NIPS*, 2009.

Kevin Gimpel and Noah A Smith. Softmax-margin CRFs: Training log-linear models with cost functions. In *Proc. of NAACL*, 2010.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*, volume 1. MIT Press Cambridge, 2016.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. In *Proc. of NIPS*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, and John Agapiou. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Peter D Grünwald and A Philip Dawid. Game theory, maximum entropy, minimum discrepancy and robust Bayesian decision theory. *Annals of Statistics*, pages 1367–1433, 2004.

Ivan Habernal and Iryna Gurevych. Argumentation mining in user-generated web discourse. *Computational Linguistics*, 2016.

Brian Harris. Bi-text, a new concept in translation theory. *Language Monthly*, 54 (March):8–10, 1988.

Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC, 2015. ISBN 1498712169, 9781498712163.

Michael Held, Philip Wolfe, and Harlan P Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

Felix Hill, KyungHyun Cho, Anna Korhonen, and Yoshua Bengio. Learning to understand phrases by embedding the dictionary. *TACL*, 4(1):17–30, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Jennifer L Hochschild and Michael Danielson. Can we desegregate public schools and subsidized housing? Lessons from the sorry history of Yonkers, New York. In Clarence Stone, editor, *Changing Urban Education*, chapter 2, pages 23–44. University Press of Kansas, Lawrence, 1998.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proc. of ICCV*, 2017.

Ozan İrsoy. *Deep Sequential and Structural Neural Models of Compositionality*. PhD thesis, Cornell University, 2017.

Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. Group lasso with overlap and graph lasso. In *Proc. of ICML*, 2009.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *Proc. of ICLR*, 2017.

Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, and Francis Bach. Proximal methods for hierarchical sparse coding. *Journal of Machine Learning Research*, 12 (Jul):2297–2334, 2011.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proc. of ICCV*, 2017.

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

Dan Jurafsky and James H Martin. *Speech and Language Processing (3rd ed.)*. draft, 2018.

Sham M Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Regularization techniques for learning with matrices. *Journal of Machine Learning Research*, 13: 1865–1890, 2012.

Arzoo Katiyar and Claire Cardie. Investigating LSTMs for joint extraction of opinion entities and relations. In *Proc. of ACL*, 2016.

Yoon Kim, Carl Denton, Loung Hoang, and Alexander M Rush. Structured attention networks. In *Proc. of ICLR*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of ICLR*, 2015.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327, 2016.

Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. OpenNMT: Open-source toolkit for neural machine translation. *arXiv e-prints*, 2017.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL*, 2007.

Lingpeng Kong, Chris Dyer, and Noah A Smith. Segmental recurrent neural networks. In *Proc. of ICLR*, 2016.

Terry Koo, Amir Globerson, Xavier Carreras Pérez, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *Proc. of EMNLP*, 2007.

Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology*. Commtext. Sage, 1980.

Rahul G Krishnan, Simon Lacoste-Julien, and David Sontag. Barrier Frank-Wolfe for marginal inference. In *Proc. of NIPS*, 2015.

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

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

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Proc. of NIPS*, 2015.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *Proc. of ICML*, 2013.

Simon Lacoste-Julien, Fredrik Lindsten, and Francis Bach. Sequential kernel herding: Frank-Wolfe optimization for particle filtering. In *Proc. of AISTATS*, 2015.

John D Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, 2001.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *Proc. of EMNLP*, 2016.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in NLP. In *Proc. of NAACL-HLT*, 2016.

Zhifei Li and Jason Eisner. First-and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proc. of EMNLP*, 2009.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *Proc. of ICLR*, 2017.

Ziheng Lin, Hwee Tou Ng, and Min-Yen Kan. A PDTB-styled end-to-end discourse parser. *Natural Language Engineering*, 20(02):151–184, 2014.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proc. of ICCVPR*, 2015.

Yang Liu and Mirella Lapata. Learning structured text representations. *TACL*, 6: 63–75, 2018.

Jeffrey S Lubbers. *A Guide to Federal Agency Rulemaking*. American Bar Association Chicago, 4th edition, 2006.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*, 2015.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proc. of ICLR*, 2017.

Jean Maillard, Stephen Clark, and Dani Yogatama. Jointly learning sentence embeddings and syntax with unsupervised tree-LSTMs. *arXiv e-prints*, 2017.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proc. of ACL (system demonstrations)*, 2014.

André FT Martins and Mariana SC Almeida. Priberam: A Turbo Semantic Parser with second order features. In *Proc. of SemEval*, 2014.

André FT Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*, 2016.

André FT Martins and Julia Kreutzer. Learning what's easy: Fully differentiable neural easy-first taggers. In *Proc. of EMNLP*, 2017.

André FT Martins, Noah A Smith, and Eric P Xing. Concise integer linear programming formulations for dependency parsing. In *Proc. of ACL-IJCNLP*, 2009.

André FT Martins, Kevin Gimpel, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. Learning structured classifiers with dual coordinate ascent. Technical report, CMU-ML-10-109, 2010.

André FT Martins, Miguel B Almeida, and Noah A Smith. Turning on the turbo: Fast third-order non-projective Turbo Parsers. In *Proc. of ACL*, 2013.

André FT Martins, Mário AT Figueiredo, Pedro MQ Aguiar, Noah A Smith, and Eric P Xing. AD3: Alternating directions dual decomposition for MAP inference in graphical models. *JMLR*, 16(1):495–545, 2015.

Ryan T McDonald and Giorgio Satta. On the complexity of non-projective data-driven dependency parsing. In *Proc. of ICPT*, 2007.

Ryan T McDonald, Koby Crammer, and Fernando CN Pereira. Online large-margin training of dependency parsers. In *Proc. of ACL*, 2005a.

Ryan T McDonald, Fernando CN Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT-EMNLP*, 2005b.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. In *Proc. of ICLR*, 2018.

Arthur Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention. In *Proc. of ICML*, 2018.

Ofer Meshi, Mehrdad Mahdavi, and Alexander G Schwing. Smooth and strong: MAP inference with linear convergence. In *Proc. of NIPS*, 2015.

Ofer Meshi, Mehrdad Mahdavi, Adrian Weller, and David Sontag. Train and test tightness of LP relaxations in structured prediction. In *Proc. of ICML*, 2016.

Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in NLP applications? In *Proc. of EMNLP*, 2016.

Andreas C Müller and Sven Behnke. PyStruct: Learning structured prediction in Python. *Journal of Machine Learning Research*, 15(1):2055–2060, 2014.

Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. DyNet: The dynamic neural network toolkit. *arXiv e-prints*, 2017.

Vlad Niculae and Mathieu Blondel. A regularized framework for sparse and structured neural attention. In *Proc. of NIPS*, 2017.

Vlad Niculae, Joonsuk Park, and Claire Cardie. Argument mining with structured SVMs and RNNs. In *Proc. of ACL*, 2017.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. SparseMAP: Differentiable sparse structured inference. In *Proc. of ICML*, 2018.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan T McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. Universal Dependencies v1: A multilingual treebank collection. In *Proc. of LREC*, 2016.

Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer New York, 1999.

Sebastian Nowozin, Peter V Gehler, Jeremy Jancsary, and Christoph H Lampert. *Advanced Structured Prediction*. MIT Press, 2014.

Guillaume Obozinski and Francis Bach. A unified perspective on convex structured sparsity: Hierarchical, symmetric, submodular norms and beyond. working paper or preprint, December 2016.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity. In *Proc. of ACL*, 2004.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proc. of EMNLP*, 2016.

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

Joonsuk Park and Claire Cardie. A corpus of eRulemaking user comments for measuring evaluability of arguments. In *Proc. of LREC*, 2018.

Joonsuk Park, Sally Klingel, Claire Cardie, Mary Newhart, Cynthia Farina, and Joan-Josep Vallbé. Facilitative moderation for online participation in eRulemaking. In *Proc. of DG.O*, 2012.

Joonsuk Park, Cheryl Blake, and Claire Cardie. Toward machine-assisted participation in eRulemaking: An argumentation model of evaluability. In *Proc. of ICAIL*, 2015a.

Joonsuk Park, Arzoo Katiyar, and Bishan Yang. Conditional Random Fields for identifying appropriate types of support for propositions in online user comments. In *Proc. of the NAACL Workshop on Argumentation Mining*, 2015b.

Andreas Peldszus and Manfred Stede. Joint prediction in MST-style discourse parsing for argumentation mining. In *Proc. of EMNLP*, 2015.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proc. of EMNLP*, 2014.

Peter Potash, Alexey Romanov, and Anna Rumshisky. Here's my point: Argumentation mining with pointer networks. *arXiv e-prints*, 2016.

PyTorch. `http://pytorch.org`, 2017.

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

Pradeep Ravikumar, Alekh Agarwal, and Martin J Wainwright. Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *JMLR*, 11:1043–1080, 2010.

Mark D Reid, Rafael M Frongillo, Robert C Williamson, and Nishant Mehta. Generalized mixability via entropic duality. In *Proc. of COLT*, 2015.

R Tyrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. Reasoning about entailment with neural attention. In *Proc. of ICLR*, 2016.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proc. of EMNLP*, 2015.

Leonard J Savage. Elicitation of personal probabilities and expectations. *Journal of the American Statistical Association*, 66(336):783–801, 1971.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241: 81–89, 2017.

Shai Shalev-Shwartz and Tong Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, 155 (1):105–145, 2016.

Morton Slater. Lagrange multipliers revisited. Cowles Foundation Discussion Papers 80, Cowles Foundation for Research in Economics, Yale University, 1959.

David A Smith and Noah A Smith. Probabilistic models of nonprojective dependency trees. In *Proc. of EMNLP*, 2007.

Noah A Smith. *Linguistic Structure Prediction*. Synth. Lect. Human Lang. Technol. Morgan & Claypool, 2011.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*, 2013.

Christian Stab and Iryna Gurevych. Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3):619–659, 2017.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. of AISTATS*, 2011.

Milan Straka and Jana Straková. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proc. of the CoNLL Shared Task on Multilingual Parsing from Raw Text to Universal Dependencies*, 2017.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured Long Short-Term Memory networks. In *Proc. of ACL-IJCNLP*, 2015.

Ben Taskar. *Learning structured prediction models: A large margin approach*. PhD thesis, Stanford University, 2004.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-Margin Markov Networks. In *Proc. of NIPS*, 2003.

Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable ADMM approach. In *Proc. of ICML*, 2016.

Lucien Tesnière. *Éléments de Syntaxe Structurale*. Librairie C. Klincksieck, 1959.

Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

Ryan J Tibshirani and Jonathan Taylor. The solution path of the generalized lasso. *Ann. Statist.*, 39(3):1335–1371, 6 2011.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proc. of ICML*, 2004.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(Sep):1453–1484, 2005.

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

Andreas Veit and Serge Belongie. Convolutional networks with adaptive computation graphs. *arXiv e-prints*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proc. of NIPS*, 2015.

Marina Vinyes and Guillaume Obozinski. Fast column generation for atomic norm regularization. In *Proc. of AISTATS*, 2017.

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

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proc. of NIPS*, 2016.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *preprint arXiv:1704.05426*, 2017.

Adina Williams, Andrew Drozdov, and Samuel R Bowman. Do latent tree learning models identify meaningful structure in sentences? *TACL*, 2018.

Robert C Williamson, Elodie Vernet, and Mark D Reid. Composite multiclass losses. *Journal of Machine Learning Research*, 2016.

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

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of ICML*, 2015.

Bishan Yang and Claire Cardie. Joint inference for fine-grained opinion extraction. In *Proc. of ACL*, 2013.

Chen Yanover and Yair Weiss. Finding the M most probable configurations using loopy belief propagation. In *Proc. of NIPS*, 2004.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. In *Proc. of ICLR*, 2017.

Yaoliang Yu. On decomposing the proximal map. In *Proc. of NIPS*, 2013.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Constantin Zălinescu. *Convex Analysis in General Vector Spaces*. World Scientific, 2002.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, et al. CoNLL 2017 shared task: Multilingual parsing from raw text to universal dependencies. *CoNLL*, 2017.

Xiangrong Zeng and Mário AT Figueiredo. Solving OSCAR regularization problems by fast approximate proximal splitting algorithms. *Digital Signal Processing*, 31:124–135, 2014.

Xiangrong Zeng and Mário AT Figueiredo. The ordered weighted ℓ_1 norm: Atomic formulation and conditional gradient algorithm. In *Proc. of SPARS*, 2015.

Leon Wenliang Zhong and James T Kwok. Efficient sparse modeling with automatic feature grouping. *IEEE transactions on neural networks and learning systems*, 23 (9):1436–1447, 2012.