Encoding and Decoding Graph Representations of Natural Language

Sam Thomson

CMU-LTI-19-002

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

Noah A. Smith *(chair)*
Taylor Berg-Kirkpatrick
Anupam Gupta
Luke Zettlemoyer

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy In Language and Information Technologies

Encoding and Decoding Graph Representations of Natural Language

Sam Thomson

CMU-LTI-19-002
March 15, 2019

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies.*

For Max and June, my angels of the morning, and Brienne, my angel of the afternoon.

Abstract

When analyzing text, it is often helpful to process the text into richer structures representing its linguistic content. Labeled directed graphs are a natural and flexible representation, especially for natural language semantics. Their generality over trees, for instance, allows them to represent relational semantics while handling phenomena like coreference and coordination.

This thesis focuses on such graphical representations of text. We develop novel algorithms and models for *decoding*—automatically extracting graphs from raw text; and *encoding*—transforming graphs into tensors so that they can be used as input to a downstream neural network. We primarily focus on *semantic* graphs—graphs that represent sentence meanings. Semantic graphs are not necessarily trees, so they require methods that can handle arbitrary graphs. We make three main contributions. First, we introduce **neural Turbo (Neurbo) parsing** and show that it is an effective approach for semantic dependency parsing, and in particular multitask semantic dependency parsing. Next, we show that the task of decoding *connected* semantic graphs can be formulated as a **Node-and-Edge-weighted Connected Subgraph (NEWCS) problem**, a novel extension of a classic, well-studied problem in network optimization called the prize-collecting Steiner tree (PCST) problem. This allows us to prove theoretical properties about decoding under connectedness constraints, and to adapt and extend existing PCST solving techniques to decoding semantic graphs. Finally, we introduce new *lawful* neural encoders whose constrained forms allow for efficient exact inference by dynamic programming. We introduce two models: **Soft Patterns (SoPa)**, which encode sequences, and **Kleene Graph Encoder Networks (KGEN)**, which encode all paths in a graph, grouped by source and destination.

Acknowledgments

I would like to sincerely thank my advisor Noah Smith. Aside from being thoughtful, smart, kind, gracious, and supremely prompt and organized, what I appreciated most about Noah as an advisor is that he always put himself in my shoes when giving advice. He shielded me from non-research concerns and gave me room to explore, and he will always be a role model for me. I would like to thank my collaborators, who taught me by example how to do this messy business that is research, which is so much more fun to do with friends: Hao Peng, Jeff Flanigan, Swabha Swayamdipta, Roy Schwartz, Chris Dyer, Nathan Schneider, Fei Liu, Meghana Kshirsagar, Brendan O'Connor, Rowan Zellers, Mark Yatskar, Yejin Choi, David Bamman, Jesse Dodge, Kenton Lee, Luke Zettlemoyer, Jaime Carbonell, Norman Sadeh, Ben Plaut, and Jan Buys. I also deeply appreciate the conversations I had with fellow students and postdocs in the ARK research group, clab, the Edvisees, the Language Technologies Institute, the Machine Learning Department, and everybody in UWNLP. I am grateful to all of the researchers whose open code and datasets made this research possible, and especially Dipanjan Das, Michael Roth, Collin Baker, Miriam Petruck, and Stephan Oepen, who provided support that went above and beyond. Thank you to everyone who graciously donated their time to provide feedback on drafts of this research at various stages, including fellow students, reviewers, and my committee. I would like to thank the professors and staff at the Language Technologies Institute, especially Lori Levin, Ed Hovy, Alon Lavie, Bob Frederking, and Stacey Young. The students, postdocs, professors and staff at the Paul G. Allen School of Computer Science & Engineering at the University of Washington welcomed me, treated me as one of their own, and made the second half of my graduate life profoundly rewarding. My high-school teacher Nancy Meinhard and my undergraduate professors Ravi Ramakrishna and Dexter Kozen sparked and developed my interest in algebra, and without them Chapter 5 would not exist. I am eternally grateful for the support of my family. My wife Brienne Thomson, my parents Rob and Lydia, my sister Eliza, and my in-laws Joey and Karen Joubert all made this possible. And finally I am grateful for my children Max and June, who provided a little extra impetus to finish this stage of my life in a timely manner.

Contents

Chapter 1

Introduction

The field of natural language processing has had a lasting love affair with trees. The community has embarked on large-scale tree-based annotation efforts, such as the Penn Treebank (PTB; Marcus et al., 1993) and the universal dependency corpus (UD; Silveira et al., 2014). And we have a swarm of algorithms and machine learning techniques at our disposal for working with linguistic structures in the shape of trees, from CKY (Cocke, 1969; Kasami, 1965; Younger, 1967), Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967), and Eisner's algorithm (Eisner, 1996), to Recursive Neural Networks (Goller and Kuchler, 1996; Costa et al., 2003; Socher et al., 2013) and TreeLSTMs (Tai et al., 2015). Trees ably represent **syntax**, the rules and process by which the words of a sentence combine with each other to form larger and larger phrases and eventually the sentence. But when one tries to represent the *result* of that combination process—the meaning or **semantics** of a sentence—one quickly finds that trees are no longer sufficient. Entities may participate in multiple events, and **graphs**, specifically labeled directed graphs, become necessary. Graphs are well-studied objects, but they have not received the same level of attention in NLP as have trees, and there remain several unanswered questions around to their use in computational linguistics. This thesis contributes several new techniques and algorithms for working with linguistic graphs. We roughly divide our inquiry into two broad research questions:

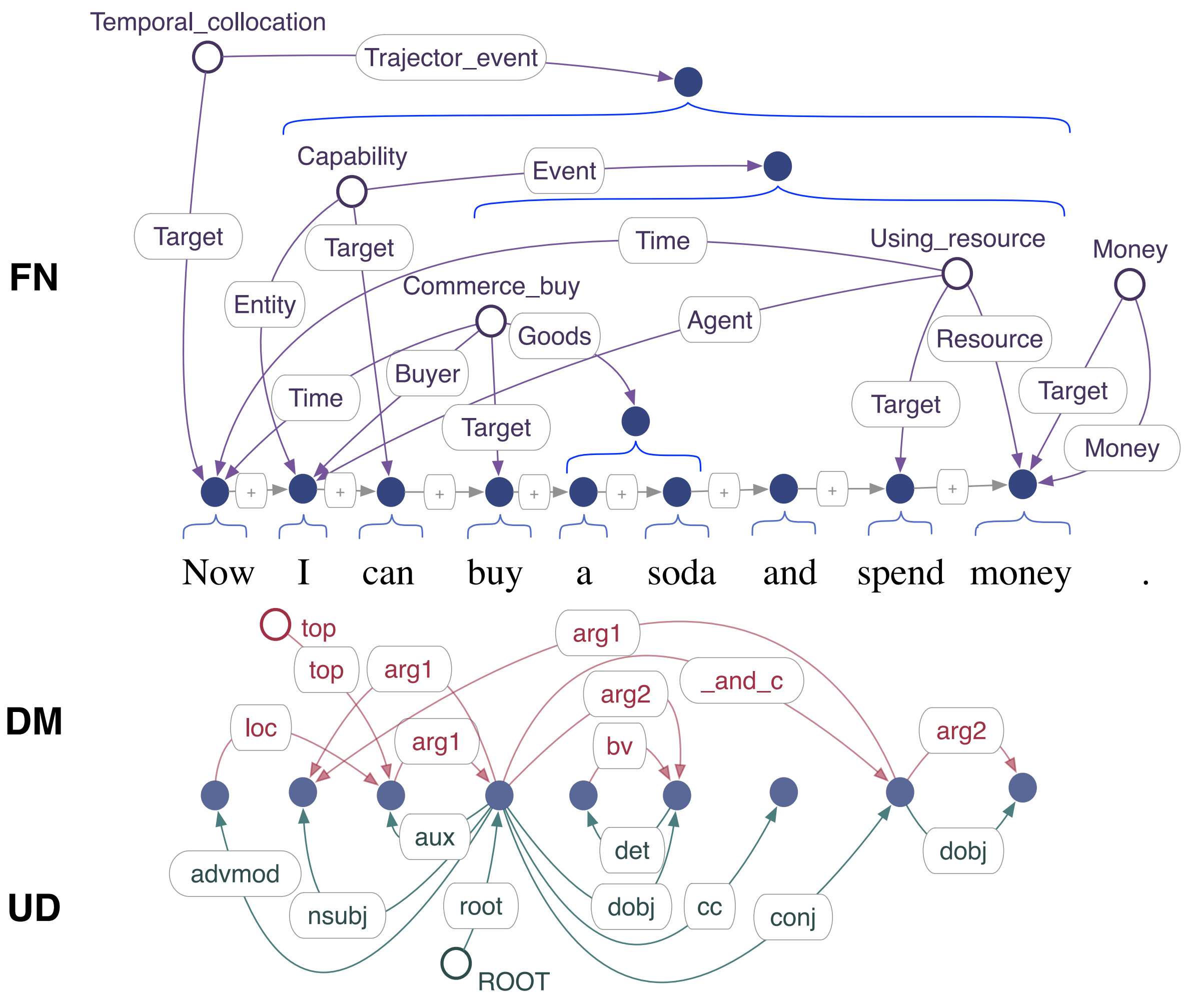


Figure 1.1: An example sentence annotated with semantic frames and roles from **FrameNet** (Baker et al., 1998, *top/purple*), semantic dependencies from **DM** (Oepen et al., 2014, *middle/red*) and syntax from **Universal Dependencies** (Silveira et al., 2014, *bottom/green*). Nodes corresponding to token spans have solid blue fill, and nodes added by a formalism are hollow. Token nodes are shown both above and below the sentence for clarity. Gray arcs labeled "+" are added between adjacent token nodes to encode the linear order of the sentence (as graphs are considered to be unordered).

- What algorithms and techniques are effective at *predicting* graphs?
- What algorithms and techniques are effective for *conditioning on* a graph (when making some other prediction)?

Following terminology first introduced by Warren Weaver in the context of machine translation, we call the act of predicting a graph **decoding**, and the act of predicting some other representation *from* a graph **encoding**.

While trees are graphs, graphs are not trees, so moving from syntax to semantics involves working with graph structures that follow *fewer* constraints. In a context-free grammar derivation (Chomsky, 1959), such as those found in PTB, each constituent (other than the sentence itself) has a single parent constituent. Once a constituent has been expanded via a production rule in the grammar, its children never interact again. Thus a syntactic parse has a neat, orderly, hierarchical shape, where constituents recursively expand until the surface words of the sentence are produced. Similarly, in a syntactic dependency parse resulting from a dependency grammar (Tesnière, 1959), each token modifies another single token, flowing all the way up to a root token, resulting in an **arborescence** (a directed graph that is a rooted tree). By contrast, a sentence's semantics is a tangled web of interactions. Specifically, the entities and concepts mentioned in a sentence may each participate in multiple events and hold multiple relationships to each other. **Coreference**, **control structure**, and **coordination** (among others), are linguistic mechanisms that can signify that a participant is involved in more than one situation. For instance, in the sentence *"Now I can buy a soda and spend money"* (Figure 1.1), *I* am the participant who has the capability referred to by *"can"*, and within that capability, *I* am also the instigating agent of both the hypothetical *buying* event and the hypothetical *spending* event.

Trees are one kind of special case of graphs, but so are other data structures used in NLP. Multiword expressions, coreference, syntax, semantics, discourse, and many more types of structured analyses are either annotated directly as a labeled directed graph (*e.g.,* Copestake et al., 2005; Hajič et al., 2012; Banarescu et al., 2013; Silveira et al., 2014), or are easily convertible to one (*e.g.,* Baker et al., 2007; Surdeanu et al., 2008). Sequences are representable as a a linear chain, which is a special case of a graph (see the gray "+" arcs in Figure 1.1, for example). Arcs of a graph have a source node and a destination node, and so naturally represent *binary* relationships, but non-binary relations (like in a logical form, for example) can be handled by the introduction of new nodes, as in neo-Davidsonian semantics (Parsons, 1990). In other words, graphs are extremely general. If our two research questions can be answered, and we can find effective graph encoders and decoders, then we have tools for working with all of these formalisms, and can use the same generic framework for any stage in a core NLP pipeline. Special cases of graphs like sequences and trees often have specialized algorithms that are more efficient, but specialized algorithms require specialized implementations, and there is definite value in developing tools that work for any or all representations out of the box. We also note that even some syntactic formalisms are moving toward non-tree graphs (de Marneffe et al., 2014). Furthermore, as we show in Chapter 3, the union of multiple linguistic analyses can be combined into a single **multigraph**, leading to improved performance. Even if the individual analyses are trees or sequences, their union will not be, and so in such multi-task learning scenarios general purpose graph algorithms are needed.

Our main interest is in single-sentence semantics, for which labeled directed graphs are an especially natural and flexible representation. Semantic graphs tend to have certain characteristics, which we tailor our algorithms toward to some extent. The nodes in a linguistic graph can be labeled, often with the name of a concept, entity, or event. Nodes can be grounded in *spans* from the sentence or they can be abstract. Arcs between nodes are labeled with a linguistic relationship that holds between the nodes, and the relationships are usually asymmetric, requiring the use of directed arcs. There tend to be $O(n)$ nodes and $O(n)$ arcs for a sentence with n tokens.[1] Semantic graphs tend to be **connected**, reflecting the fact that sentences often express a single *coherent* statement or question. See Figure 1.1 for an example sentence with several kinds of annotations.

[1]With this in mind, $O(n^3)$ algorithms are often acceptable (as in Section 5.3), but NP-hard problems (as in Chapter 4) are not, and require approximate algorithms.

This thesis makes four main contributions.

- First, in Chapter 2, we introduce an arc-factored model for **semantic dependency parsing** (**SDP**; Oepen et al., 2014, 2015). We explore a variety of features and graph constraints, doing extensive empirical comparisons to find practical model choices for each of the three SDP formalisms.
- In Chapter 3, we introduce **Neural Turbo Parsing (NeurboParsing)** and show that it is an even more effective approach for semantic dependency parsing. Turbo parsing allows for decoding with higher-order factors that score small motifs in the output graph. In particular we show how NeurboParsing can be used for *multitask* semantic dependency parsing, where we get state-of-the-art results in three SDP formalisms.
- Next (Chapter 4), we show that the problem of decoding *connected* semantic graphs is an extension of a classic, well-studied problem in network optimization called the **prize-collecting Steiner tree (PCST)** problem (Goemans and Williamson, 1992). This allows us to prove theoretical properties about decoding under certain graph constraints, and to adapt and extend existing PCST solving techniques to decoding semantic graphs.
- In Chapter 5, we introduce two new *lawful* neural encoders, **Soft Patterns** (**SoPa**) and **Kleene Graph Encoder Networks** (**KGEN**) whose constrained forms allow for efficient exact inference. SoPa encodes *sequences* using an array of small neurally-weighted finite state automata, and we test it on text classification. KGEN extends SoPa in order to encode *graphs*. The resulting output of KGEN is as if every path in a graph had been encoded with a SoPa-like sequence model, then grouped by source and destination, then max-pooled. Even though there are infinitely many paths in a graph, this can be done efficiently with dynamic programming.

Chapters 2–4 work toward better graph decoding, and Chapter 5 is devoted to graph encoding. These contributions together are aimed toward a generic *graph-to-graph* pipeline for structured prediction. Given any task where one must predict some structured output $y \in \mathcal{Y}$ given some structured input $x \in \mathcal{X}$:

1. convert x and y to multigraphs G_x and G_y, respectively,
2. **encode** x as a contextualized tensor v_x, and
3. **decode** y, conditioned on v_x.

While not the only options, we demonstrate in this thesis that KGEN is an effective method for step 2, and that NeurboParsing is an effective approach for step 3. Our experiments are tailored toward broad-coverage semantic dependency parsing, but our methods have much broader potential impact because they apply generally to graphs.

Chapter 2

Arc-factored Semantic Dependency Parsing

[*] Semantic dependencies are a form of semantic graph representation where the tokens of a sentence themselves are the nodes of the graph. The arcs represent semantic relations between all content-bearing words in a sentence. In this chapter, we explore how varying sets of features and constraints affect semantic dependency parsing performance in a linear, arc-factored model.

We present in this chapter two systems that produce semantic dependency parses in the three formalisms of the SemEval Shared Tasks on Broad-Coverage Semantic Dependency Parsing (Oepen et al., 2014, 2015). We describe the task and data in §2.1. These systems generate parses by extracting features for each potential dependency arc and learning a statistical model to discriminate between good arcs and bad; the first treats each labeled edge decision as an independent multiclass logistic regression (§2.2.2), while the second predicts arcs as part of a graph-based structured support vector machine (§2.2.2). Common to both models is a rich set of features on arcs, described in §2.2.2. We include a discussion of features found to have no discernible effect, or negative effect, during development (§2.4).

Our system placed second in the open track of the SemEval 2014 Shared Task 8 on Broad-

[*] *The work described in this chapter was done with Brendan O'Connor, Jeffrey Flanigan, David Bamman, Jesse Dodge, Swabha Swayamdipta, Nathan Schneider, Chris Dyer, and Noah A. Smith, and published as Thomson et al. (2014).*

	DM		**PAS**		**PSD**	
	id	ood	id	ood	id	ood
# labels	59	47	42	41	91	74
% trees	2.3	9.7	1.2	2.4	42.2	51.4
% projective	2.9	8.8	1.6	3.5	41.9	54.4

Table 2.1: Graph statistics for in-domain (WSJ, "id") and out-of-domain (Brown corpus, "ood") data. Numbers taken from Oepen et al. (2015).

Coverage Semantic Dependency Parsing task (in which output from syntactic parsers and other outside resources *can* be used). We present our results in §2.3. In Chapter 3, we improve on this work by showing that the multiple semantic dependency formalisms can be decoded *jointly*, with higher-order neural factors that model the formalisms' correlations with each other. This multitask learning approach gives us state-of-the art performance on all three formalisms in the SDP shared task.

2.1 Broad-Coverage Semantic Dependency Parsing (SDP)

First defined in a SemEval 2014 shared task (Oepen et al., 2014), and then extended by Oepen et al. (2015), the broad-coverage semantic dependency parsing (**SDP**) task is centered around three semantic formalisms whose annotations have been converted into bilexical dependencies. Broad-coverage semantic dependency parsing aims to provide a shallow semantic analysis of text not limited to a specific application domain. As distinct from deeper semantic analysis (e.g., parsing to a full lambda-calculus logical form), shallow semantic parsing captures relationships between pairs of words or concepts in a sentence, and has wide application for information extraction, knowledge base population, and question answering (among others). See Figure 2.1 for an example. The formalisms come from varied linguistic traditions, but all three aim to capture predicate-argument relations between content-bearing words in a sentence.

While at first glance similar to syntactic dependencies, semantic dependencies have distinct goals and characteristics, more akin to semantic role labeling (SRL; Gildea and Jurafsky, 2002) or the abstract meaning representation (AMR; Banarescu et al., 2013). They abstract over different syntactic realizations of the same or similar meaning (e.g., *"She gave me the ball."* vs. *"She gave the ball to me."*). Conversely, they attempt to distinguish between different senses even when realized in similar syntactic forms (e.g., *"I baked in the kitchen."* vs. *"I baked in the sun."*).

Structurally, they are labeled directed graphs whose vertices are tokens in the sentence. This is in contrast to AMR whose vertices are abstract concepts, with no explicit alignment to tokens, which makes parsing more difficult (Flanigan et al., 2014). Their arc labels encode broadly-applicable semantic relations rather than being tailored to any specific downstream application or ontology.[1] They are not necessarily trees, because a token may be an argument of more than one predicate (e.g., in *"John wants to eat,"* John is both the wanter and the would-be eater). Their analyses may optionally leave out non–content-bearing tokens, such as punctuation or the infinitival *"to,"* or prepositions that simply mark the type of relation holding between other words. But when restricted to content-bearing tokens (including adjectives, adverbs, etc.), the subgraph is connected. In this sense, SDP provides a *whole-sentence* analysis. This is in contrast to PropBank-style SRL, which gives an analysis of only verbal and nominal predicates (Palmer et al., 2005). Semantic dependency graphs also tend to have higher levels of non-projectivity than syntactic trees (Oepen et al., 2014). Sentences with graphs containing cycles have been removed from the dataset by the organizers, so all remaining graphs are directed acyclic graphs. The three formalisms also have "top" annotations, corresponding roughly to the semantic focus of the sentence. A "top" need not be a root of the graph. Because they have the same format and follow roughly the same constraints, this allows us to use the same machinery (§2.2) for training and testing statistical models for the three formalisms. We will see in the next chapter that it also makes the tasks amenable to multitask learning. Table 2.1 summarizes some of the dataset's high-level statistics.

[1]This may make another disambiguation step necessary to use these representations in a downstream task, but there is evidence that modeling semantic composition separately from grounding in any ontology is an effective way to achieve broad coverage (Kwiatkowski et al., 2013).

Formalisms. Following the SemEval shared tasks, we consider three formalisms. The **DM** (DELPH-IN MRS) representation comes from DeepBank (Flickinger et al., 2012), which are manually-reranked parses from the LinGO English Resource Grammar (Copestake and Flickinger, 2000). LinGO is a head-driven phrase structure grammar (HPSG; Pollard and Sag, 1994) whose syntactic derivations simultaneously produce MRS graphs. The **PAS** (Predicate-Argument Structures) representation is extracted from the Enju Treebank, which consists of automatic parses from the Enju HPSG parser (Miyao, 2006). PAS annotations are also available for the Penn Chinese Treebank (Xue et al., 2005). The **PSD** (Prague Semantic Dependencies) representation is extracted from the tectogrammatical layer of the Prague Czech-English Dependency Treebank (Hajič et al., 2012). PSD annotations are also available for a Czech translation of the WSJ Corpus. In this work, we train and evaluate only on English annotations. Of the three, PAS follows syntax most closely, and prior work has found it the easiest to predict. PSD has the largest set of labels, and parsers have significantly lower performance on it (Oepen et al., 2015).

2.2 Models

We treat the problem as a three-stage pipeline. The first stage prunes words by predicting whether they have any incoming or outgoing edges at all (§2.2.1); if a word does not, then it is not considered for any attachments in later stages. The second stage predicts where edges are present, and their labels (§2.2.2). The third stage predicts whether a predicate word is a *top* or not (§2.2.3). Formalisms sometimes annotate more than one "top" per sentence, but we found that we achieve the best performance on all formalisms by predicting only the one best-scoring "top" under the model.

2.2.1 Singleton Classification

For each formalism, we train a classifier to recognize *singletons*, nodes that have no parents or children. (For example, punctuation tokens are often singletons.) This makes the system faster without affecting accuracy. For singleton prediction, we use a token-level logistic regression classifier, with features including the word, its lemma, and its part-of-speech tag. If the classifier predicts a probability of 99% or higher the token is pruned; this removes around 10% of tokens. (The classifier performs differently on different formalisms; on PAS it has perfect accuracy, while on DM and PSD accuracy is in the mid-90's.)

2.2.2 Edge Prediction

In the second stage of the pipeline, we predict the set of labeled directed edges in the graph. We use the same set of edge-factored features (§2.2.2) in two alternative models: an edge-independent multiclass logistic regression model (LOGISTICEDGE, §2.2.2); and a structured SVM (Taskar et al., 2003; Tsochantaridis et al., 2004) that enforces a *determinism* constraint for certain labels, which allows each word to have at most one outgoing edge with that label (SVMEDGE, §2.2.2). For each formalism, we trained both models with varying features enabled and hyperparameter settings and submitted the configuration that produced the best labeled F_1 on the development set. For DM and PSD, this was LOGISTICEDGE; for PAS, this was SVMEDGE. We report results only for the submitted configurations, with different features enabled. Due to time constraints, full hyperparameter sweeps and comparable feature sweeps were not possible.

LOGISTICEDGE Parser

The LOGISTICEDGE model considers only token index pairs (i, j) where $|i - j| \leq 10$, $i \neq j$, and both t_i and t_j have been predicted to be non-singletons by the first stage. Although this prunes some gold edges, among the formalisms, 95%–97% of all gold edges are between tokens of distance 10 or less. Both directions $i \rightarrow j$ and $j \rightarrow i$ are considered between every pair.

Let L be the set of $K + 1$ possible output labels: the formalism's original K edge labels, plus the additional label NOEDGE, which indicates that no edge exists from i to j. The model treats every pair of token indices (i, j) as an independent multiclass logistic regression over output space

L. Let x be an input sentence. For candidate parent index i, child index j, and edge label ℓ, we extract a feature vector $\boldsymbol{f}(x, i, j, \ell)$, where ℓ is conjoined with every feature described in §2.2.2. The multiclass logistic regression model defines a distribution over L, parameterized by weights $\boldsymbol{\phi}$:

$$P(\ell \mid \boldsymbol{\phi}, x, i, j) = \frac{\exp\{\boldsymbol{\phi} \cdot \boldsymbol{f}(x, i, j, \ell)\}}{\sum_{\ell' \in L} \exp\{\boldsymbol{\phi} \cdot \boldsymbol{f}(x, i, j, \ell')\}}. \tag{2.1}$$

$\boldsymbol{\phi}$ is learned by minimizing total negative log-likelihood of the above (with weighting; see below), plus ℓ_2 regularization. AdaGrad (Duchi et al., 2011) is used for optimization. This seemed to optimize faster than L-BFGS (Liu and Nocedal, 1989), at least for earlier iterations, though we did no systematic comparison. Stochastic gradient steps are applied one at a time from individual examples, and a gradient step for the regularizer is applied once per epoch.

The output labels have a class imbalance; in all three formalisms, there are many more NoEdge examples than true edge examples. We improved F_1 performance by downweighting NoEdge examples through a weighted log-likelihood objective, $\sum_{i,j} \sum_{\ell} w_\ell \log P(\ell \mid \boldsymbol{\phi}, x, i, j)$, with $w_{\text{NoEdge}} = 0.3$ (selected on development set) and $w_\ell = 1$ otherwise.

Decoding: To predict a graph structure at test-time for a new sentence, the most likely edge label is predicted for every candidate (i, j) pair of unpruned tokens. If an edge is predicted for both directions for a single (i, j) pair, only the edge with the higher score is chosen. (There are no such bidirectional edges in the training data.) This post-processing actually did not improve accuracy on DM or PSD; it did improve PAS by ≈0.2% absolute F_1, but we did not submit LogisticEdge for PAS.

SvmEdge Parser

In the SvmEdge model, we use a structured SVM with a determinism constraint. This constraint ensures that each word token has at most one outgoing edge for each label in a set of deterministic labels L_d. For example, in DM a predicate never has more than one child with edge label "ARG1."

L_d was chosen to be the set of edges that were $> 99.9\%$ deterministic in the training data.[2]

Consider the fully dense graph of all edges between all words predicted as not singletons by the singleton classifier §2.2.1 (in all directions with all possible labels). Unlike LOGISTICEDGE, the label set L does not include an explicit NOEDGE label. If $\boldsymbol{\psi}$ denotes the model weights, and $\boldsymbol{f}$ denotes the features, then an edge from i to j with label ℓ in the dense graph has a weight $c(i, j, \ell)$ assigned to it using the linear scoring function $c(i, j, \ell) = \boldsymbol{\psi} \cdot \boldsymbol{f}(x, i, j, \ell)$.

Decoding: For each node and each label ℓ, if $\ell \in L_d$, the decoder adds the highest scoring outgoing edge, if its weight is positive. For $\ell \notin L_d$, every outgoing edge with positive weight is added. This procedure is guaranteed to find the highest scoring subgraph (largest sum of edge weights) of the dense graph subject to the determinism constraints. Its runtime is $O(n^2)$.

The model weights are trained using the **structured hinge loss** (Tsochantaridis et al., 2004). If x is a sentence and y is a graph over that sentence, let the features be denoted $\boldsymbol{f}(x, y) = \sum_{(i,j,\ell) \in y} \boldsymbol{f}(x, i, j, \ell)$. The structured hinge loss for each training example (x_i, y_i) is:

$$-\boldsymbol{\psi}^\top \boldsymbol{f}(x_i, y_i) + \max_y \{\boldsymbol{\psi}^\top \boldsymbol{f}(x_i, y) + \mathit{cost}(y, y_i)\} \tag{2.2}$$

where $\mathit{cost}(y, y_i) = \alpha|y \setminus y_i| + \beta|y_i \setminus y|$. α and β trade off between precision and recall for the edges (Gimpel and Smith, 2010). The loss is minimized with AdaGrad using early-stopping on a development set.

Edge Features

Table 2.2 describes the features we used for predicting edges. These features were computed over an edge e with parent token s at index i and child token t at index j. Unless otherwise stated, each feature template listed has an indicator feature that fires for each value it can take on. For the

[2]By this we mean that of the nodes that have at least one outgoing ℓ edge, 99.9% of them have only one outgoing ℓ edge. For DM, $L_d = L \backslash \{$"_and_c," "_or_c," "_then_c," "loc," "mwe," "subord"$\}$; for PAS, $L_d = L$; and for PSD, $L_d = \{$"DPHR," "INTF," "VOCAT"$\}$.

submitted results, LogisticEdge uses all features except Dependency Path v2, POS Path, and Distance Thresholds, and SvmEdge uses all features except Dependency Path v1. This was due to SvmEdge being faster to train than LogisticEdge when including POS Path features, and due to time constraints for the submission we were unable to retrain LogisticEdge with these features.

Feature Hashing

The biggest memory usage was in the map from feature names to integer indices during feature extraction. For experimental expedience, we implemented multitask feature hashing (Weinberger et al., 2009), which hashes feature names to indices, under the theory that errors due to collisions tend to cancel. No drop in accuracy was observed.

2.2.3 Top Prediction

We trained a separate token-level binary logistic regression model to classify whether a token's node had the "top" attribute or not. At decoding time, all predicted predicates (i.e., nodes where there is at least one outbound edge) are possible candidates to be "top"; the classifier probabilities are evaluated, and the highest-scoring node is chosen to be "top." This is suboptimal, since some graphs have multiple tops (in PSD this is more common); but selection rules based on probability thresholds gave worse F_1 performance on the dev set. For a given token t at index i, the top classifier's features included t's POS tag, i, those two conjoined, and the depth of t in the syntactic dependency tree.

2.3 Experiments and Results

We participated in the Open Track, and used the syntactic dependency parses supplied by the organizers. Feature engineering was performed on a development set (§20), training on §§00–19. We evaluate labeled precision (LP), labeled recall (LR), labeled F_1 (LF), and labeled whole-sentence

Tokens: The tokens s and t themselves.

Lemmas: Lemmas of s and t.

POS tags: Part of speech tags of s and t.

Linear Order: Fires if $i < j$.

Linear Distance: $i - j$.

Dependency Path v1 (LOGISTICEDGE only): The concatenation of all POS tags, arc labels and up/down directions on the path in the syntactic dependency tree from s to t. Conjoined with s, with t, and without either.

Dependency Path v2 (SVMEDGE only): Same as Dependency Path v1, but with the lemma of s or t instead of the word, and substituting the token for any "IN" POS tag.

Up/Down Dependency Path: The sequence of upward and downward moves needed to get from s to t in the syntactic dependency tree.

Up/Down/Left/Right Dependency Path: The unlabeled path through the syntactic dependency tree from s to t, annotated with whether each step through the tree was up or down, and whether it was to the right or left in the sentence.

Is Parent: Fires if s is the parent of t in the syntactic dependency parse.

Dependency Path Length: Distance between s and t in the syntactic dependency parse.

POS Context: Concatenated POS tags of tokens at $i-1$, i, $i+1$, $j-1$, j, and $j+1$. Concatenated POS tags of tokens at $i-1, i, j-1$, and j. Concatenated POS tags of tokens at $i, i+1, j$, and $j+1$.

Subcategorization Sequence: The sequence of dependency arc labels out of s, ordered by the index of the child. Distinguish left children from right children. If t is a direct child of s, distinguish its arc label with a "+". Conjoin this sequence with the POS tag of s.

Subcategorization Sequence with POS: As above, but add the POS tag of each child to its arc label.

POS Path (SVMEDGE only): Concatenated POS tags between and including i and j. Conjoined with head lemma, with dependent lemma, and without either.

Distance Thresholds (SVMEDGE only): Fires for every integer between 1 and $\lfloor \log(|i-j|+1)/\log(1.39) \rfloor$ inclusive.

Table 2.2: Features used in edge prediction

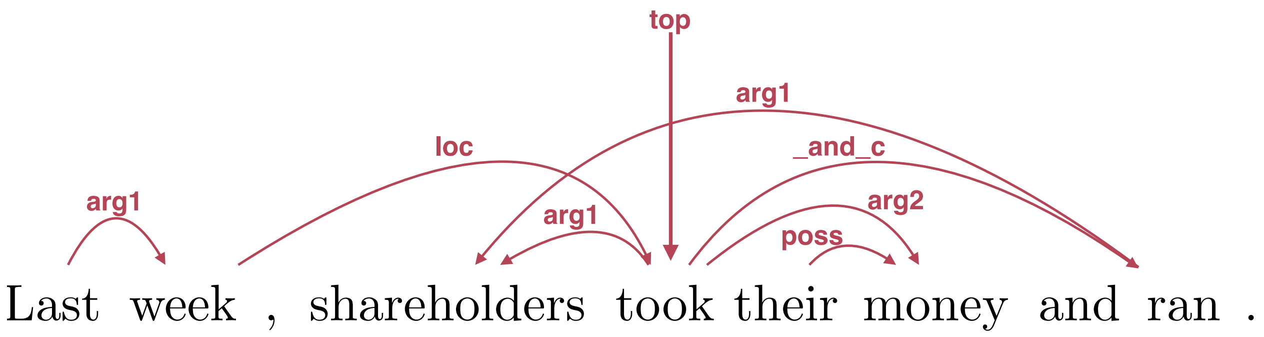
	LP	LR	LF	LM
DM	0.8446	0.8348	0.8397	0.0875
PAS	0.9078	0.8851	0.8963	0.2604
PSD	0.7681	0.7072	0.7364	0.0712
Average	0.8402	0.8090	0.8241	0.1397

Table 2.3: Labeled precision (LP), recall (LR), F_1 (LF), and whole-sentence match (LM) on the held-out (in-domain) test data.

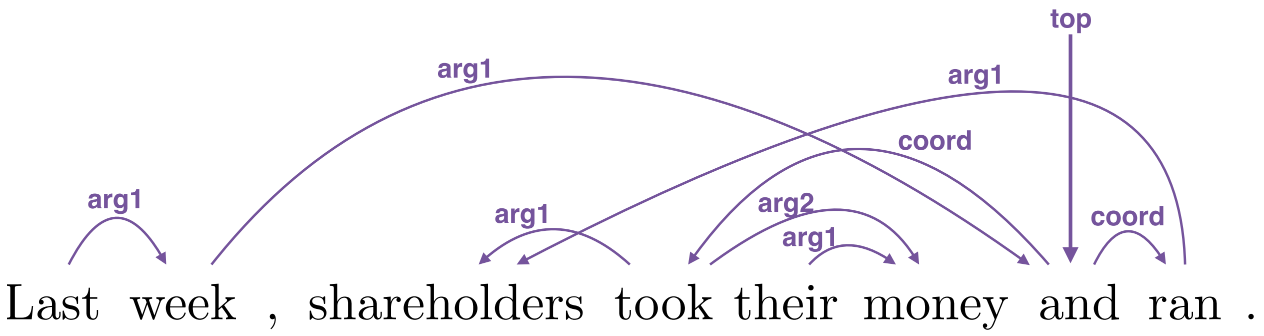
	DM	PAS	PSD	Avg.
CMU (ours)	0.8397	0.8963	0.7364	0.8241
M&A	**0.8916**	**0.9176**	**0.7790**	**0.8627**

Table 2.4: Labeled F_1 on the held-out (in-domain) test data.

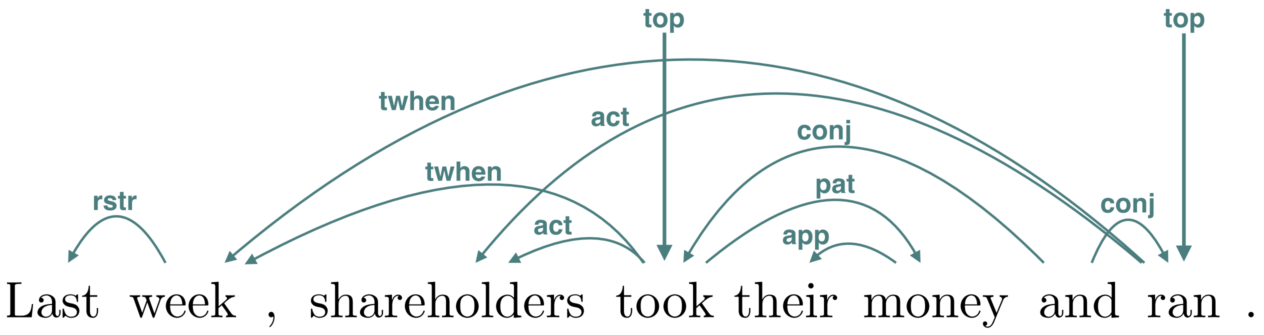
match (LM) on the held-out test data using the evaluation script provided by the organizers. LF was averaged over the formalisms to determine the winning system. Table 2.3 shows a detailed breakdown of our scores. In Table 2.4, we compare our labeled F_1 to that of the winning system, due to Martins and Almeida (2014). The winning system took a graph-based approach similar to ours. Their main improvement over our system was a set of *second-order* features, used to score adjacent arcs, with inference via AD^3 (Martins et al., 2011). They resubmitted a similar system to the 2015 version of the shared task. In the next chapter, we build off of their 2015 model.



(a) DM



(b) PAS



(c) PSD

Figure 2.1: An example sentence annotated with the three semantic formalisms of the broad-coverage semantic dependency parsing shared tasks.

Word vectors: Features derived from 64-dimensional vectors from (Faruqui and Dyer, 2014), including the concatenation, difference, inner product, and element-wise multiplication of the two vectors associated with a parent-child edge. We also trained a Random Forest on the word vectors using Liaw and Wiener's (2002) *R* implementation. The predicted labels were then used as features in LOGISTICEDGE.
Brown clusters: Features derived from Brown clusters (Brown et al., 1992) trained on a large corpus of web data. Parent, child, and conjoined parent-child edge features from cluster prefixes of length 2, 4, 6, 8, 10, and 12. Conjunctions of those features with the POS tags of the parent and child tokens.
Active/passive: Active/passive voice feature (as in Johansson and Nugues (2008)) conjoined with both the Linear Distance features and the Subcategorization Sequence features. Voice information may already be captured by features from the Stanford dependency–style parses, which include passivization information in arc labels such as *nsubjpass* and *auxpass* (de Marneffe and Manning, 2008).
Connectivity constraint: Enforcing that the graph is connected (ignoring singletons), similar to Flanigan et al. (2014). Almost all semantic dependency graphs in the training data are connected (ignoring singletons), but we found that enforcing this constraint significantly hurt precision.
Tree constraint: Enforces that the graph is a tree. Unsurprisingly, as most graphs are not trees, we found that enforcing a tree constraint hurt performance.

Table 2.5: Features and constraints giving negative results.

2.4 Negative Results

We followed a forward-selection process during feature engineering. For each potential feature, we tested the current feature set versus the current feature set plus the new potential feature. If the new feature did not improve performance, we did not add it. We list in table 2.5 some of the features which we tested but did not improve performance.

In order to save time, we ran these feature selection experiments on a subsample of the training data, for a reduced number of iterations. These results thus come with the strong caveat that the experiments were not exhaustive. It may be that some of these features could help under more careful study.

2.5 Conclusion

We found that feature-rich discriminative models perform well at the task of mapping from sentences to semantic dependency parses. We also noted additional features and constraints which did not appear to help (contrary to expectation). There are a number of clear extensions to this work that could potentially improve performance. While an edge-factored model allows for efficient inference, there is much to be gained from **higher-order features** (McDonald and Pereira, 2006; Martins et al., 2013). The amount of information shared between the three formalisms suggests that a **multitask learning** (Evgeniou and Pontil, 2004) framework could lead to gains. In fact, in the following chapter we introduce an improved model that explores both of these ideas, additionally replacing hand-engineered features with recurrent neural factors. Also, there is structure in the formalisms which we did not exploit (such as the deterministic processes by which an original PCEDT tree annotation was converted into a graph); formulating more subtle **graph constraints** to capture this a priori knowledge could lead to improved performance.

Chapter 3

Neural Turbo Semantic Parsing

[*] In this chapter, we hypothesize that the overlap among theories and their corresponding representations can be exploited using multitask learning (Caruana, 1997), allowing us to learn from more data. We again use the SemEval shared task on Broad-Coverage Semantic Dependency Parsing (**SDP**; Oepen et al., 2015) as our testbed. The shared task provides an English-language corpus with parallel annotations for three semantic graph representations, described in §2.1. Though the shared task was designed in part to encourage comparison between the formalisms, we are the first to treat SDP as a multitask learning problem.

As a strong baseline, we introduce a new neural system that parses each formalism separately (§3.1). It uses a bidirectional-LSTM composed with a multi-layer perceptron to score arcs and predicates, and has efficient, nearly arc-factored inference. Experiments show it significantly improves on state-of-the-art methods (§3.1.4).

We then present two multitask extensions (§3.2.2 and §3.2.3), with a parameterization and factorization that implicitly models the relationship between multiple formalisms. Experiments show that both techniques improve over our basic model, with an additional (but smaller) improvement when they are combined (§3.2.4). Our analysis shows that the improvement in unlabeled F_1 is

[*] *The work described in this chapter was done with Hao Peng and Noah A. Smith, and published as Peng et al. (2017a).*

greater for the two formalisms that are more structurally similar, and suggests directions for future work. Finally, we survey related work (§3.3), and summarize our contributions and findings (§3.4).

3.1 Single-Task SDP

Here we introduce our basic model, in which training and prediction for each formalism is kept completely separate, as in Chapter 2. We also lay out basic notation, which will be reused for our multitask extensions.

3.1.1 Problem Formulation

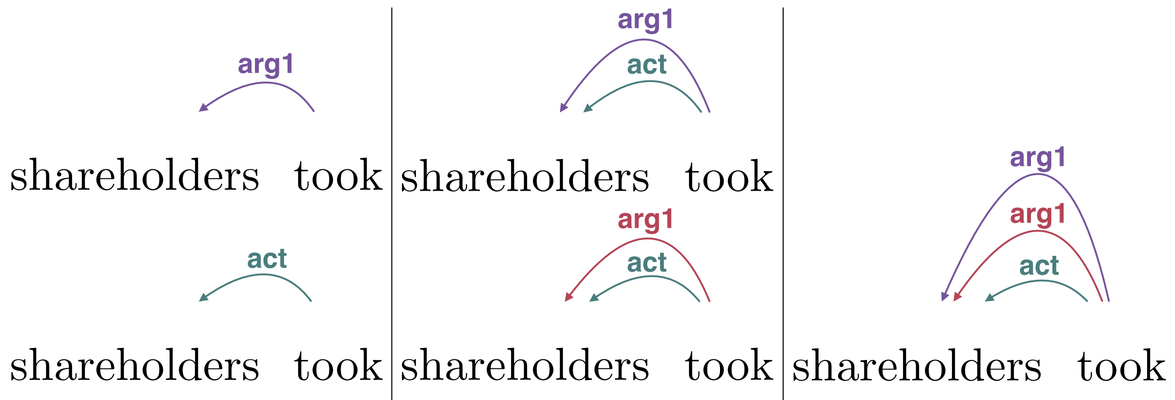
The output of semantic dependency parsing is a labeled directed graph (see Figure 2.1). Each arc has a label from a predefined set $\mathcal{L}$, indicating the semantic relation of the child to the head. Given input sentence x, let $\mathcal{Y}(x)$ be the set of possible semantic graphs over x. The graph we seek maximizes a score function S:

$$\hat{y} = \arg\max y \in \mathcal{Y}(x) S(x, y), \tag{3.1}$$

We decompose S into a sum of local scores s for **local structures** (or "parts") p in the graph:

$$S(x, y) = \sum_{p \in y} s(p). \tag{3.2}$$

For notational simplicity, we omit the dependence of s on x. See Figure 3.1a for examples of local structures. s is a parameterized function, whose parameters (denoted Θ and suppressed here for clarity) will be learned from the training data (§3.1.3). Since we search over every possible labeled graph (i.e., considering each labeled arc for each pair of words), our approach can be considered a **graph-based** (or **all-pairs**) method. The models presented in this work all share this common



(a) First-order. **(b)** Second-order. **(c)** Third-order.

Figure 3.1: Examples of local structures. We refer to the number of arcs that a structure contains as its **order**.

graph-based approach, differing only in the set of structures they score and in the parameterization of the scoring function s. This approach also underlies state-of-the-art approaches to SDP (Martins and Almeida, 2014).

3.1.2 Basic Model

Our basic model is inspired by recent successes in neural arc-factored graph-based dependency parsing (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017; Kuncoro et al., 2016). It borrows heavily from the neural arc-scoring architectures in those works, but decodes with a different algorithm under slightly different constraints.

Basic Structures

Our basic model factors over three types of structures (p in Equation 3.2):

- predicate, indicating a predicate word, denoted $i{\rightarrow}\cdot$;
- unlabeled arc, representing the existence of an arc from a predicate to an argument, denoted $i{\rightarrow}j$;
- labeled arc, an arc labeled with a semantic role, denoted $i \xrightarrow{\ell} j$.

Here i and j are word indices in a given sentence, and ℓ indicates the arc label. This list corresponds to the most basic structures used by Martins and Almeida (2014). Selecting an output y corresponds precisely to selecting which instantiations of these structures are included.

To ensure the internal consistency of predictions, the following constraints are enforced during decoding:

- $i \rightarrow \cdot$ if and only if there exists at least one j such that $i \rightarrow j$;
- If $i \rightarrow j$, then there must be exactly one label ℓ such that $i \xrightarrow{\ell} j$. Conversely, if not $i \rightarrow j$, then there must not exist any $i \xrightarrow{\ell} j$;

We also enforce a **determinism** constraint (Flanigan et al., 2014): certain labels must not appear on more than one arc emanating from the same token. The set of deterministic labels is decided based on their appearance in the training set. Notably, we do not enforce that the predicted graph is connected or spanning. If not for the predicate and determinism constraints, our model would be **arc-factored**, and decoding could be done for each i, j pair independently. Our structures do overlap though, and we employ AD^3 (Martins et al., 2011) to find the highest-scoring internally consistent semantic graph. AD^3 is an approximate discrete optimization algorithm based on dual decomposition. It can be used to decode factor graphs over discrete variables when scored structures overlap, as is the case here.

Basic Scoring

Similarly to Kiperwasser and Goldberg (2016), our model learns representations of tokens in a sentence using a bi-directional LSTM (BiLSTM). Each different type of structure (predicate, unlabeled arc, labeled arc) then shares these same BiLSTM representations, feeding them into a multilayer perceptron (MLP) which is specific to the structure type. We present the architecture slightly differently from prior work, to make the transition to the multitask scenario (§3.2) smoother. In our presentation, we separate the model into a function ϕ that represents the input (corresponding to the BiLSTM and the initial layers of the MLPs), and a function ψ that represents the output (corresponding to the final layers of the MLPs), with the scores given by their inner product.[1]

Distributed input representations. Long short-term memory networks (LSTMs) are a variant of recurrent neural networks (RNNs) designed to alleviate the vanishing gradient problem in RNNs (Hochreiter and Schmidhuber, 1997). A bi-directional LSTM (BiLSTM) runs over the sequence in both directions (Schuster and Paliwal, 1997; Graves, 2012).

Given an input sentence x and its corresponding part-of-speech tag sequence, each token is mapped to a concatenation of its word embedding vector and POS tag vector. Two LSTMs are then run in opposite directions over the input vector sequence, outputting the concatenation of the two hidden vectors at each position i: $\mathbf{h}_i = \left[\overrightarrow{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_i\right]$ (we omit $\mathbf{h}_i$'s dependence on x and its own parameters). $\mathbf{h}_i$ can be thought of as an encoder that contextualizes each token conditioning on all of its context, without any Markov assumption. $\mathbf{h}$'s parameters are learned jointly with the rest of the model (§3.1.3); we refer the readers to Cho (2015) for technical details.

The input representation $\boldsymbol{\phi}$ of a predicate structure depends on the representation of one word:

$$\boldsymbol{\phi}(i\rightarrow\cdot) = \tanh\left(\mathbf{C}_{\text{pred}}\mathbf{h}_i + \mathbf{b}_{\text{pred}}\right). \tag{3.3a}$$

For unlabeled arc and labeled arc structures, it depends on both the head and the modifier (but not the label, which is captured in the distributed output representation):

$$\boldsymbol{\phi}(i\rightarrow j) = \tanh\left(\mathbf{C}_{\text{UA}}\left[\mathbf{h}_i; \mathbf{h}_j\right] + \mathbf{b}_{\text{UA}}\right), \tag{3.3b}$$

$$\boldsymbol{\phi}(i \xrightarrow{\ell} j) = \tanh\left(\mathbf{C}_{\text{LA}}\left[\mathbf{h}_i; \mathbf{h}_j\right] + \mathbf{b}_{\text{LA}}\right). \tag{3.3c}$$

Distributed output representations. NLP researchers have found that embedding discrete output labels into a low dimensional real space is an effective way to capture commonalities among them (*e.g.,* Srikumar and Manning, 2014; Hermann et al., 2014; FitzGerald et al., 2015). In neural

[1]For clarity, we present single-layer BiLSTMs and MLPs, while in practice we use two layers for both.

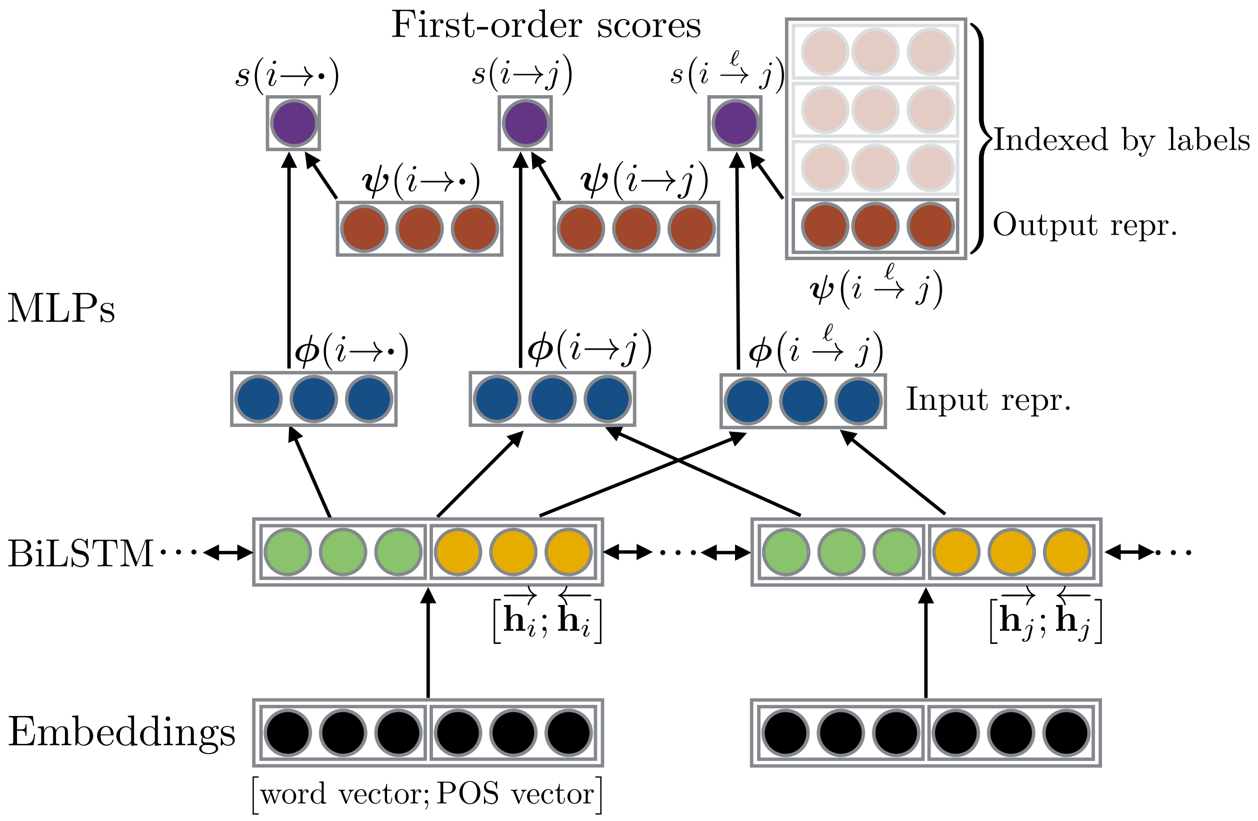

First-order scores
$s(i\to\cdot)$
$s(i\to j)$
$s(i\xrightarrow{\ell} j)$
Indexed by labels
$\boldsymbol{\psi}(i\to\cdot)$
$\boldsymbol{\psi}(i\to j)$
Output repr.
$\boldsymbol{\psi}(i\xrightarrow{\ell} j)$
MLPs
$\boldsymbol{\phi}(i\to\cdot)$
$\boldsymbol{\phi}(i\to j)$
$\boldsymbol{\phi}(i\xrightarrow{\ell} j)$
Input repr.
BiLSTM
$[\overrightarrow{\mathbf{h}}_i;\overleftarrow{\mathbf{h}}_i]$
$[\overrightarrow{\mathbf{h}}_j;\overleftarrow{\mathbf{h}}_j]$
Embeddings
[word vector; POS vector]

Figure 3.2: Illustration of the architecture of the basic model. i and j denote the indices of tokens in the given sentence. The figure depicts single-layer BiLSTM and MLPs, while in practice we use two layers for both.

language models (*e.g.*, Bengio et al., 2003; Mnih and Hinton, 2007) the weights of the output layer could also be regarded as an output embedding.

We associate each first-order structure p with a d-dimensional real vector $\boldsymbol{\psi}(p)$ which does not depend on particular words in p. Predicates and unlabeled arcs are each mapped to a single vector:

$$\boldsymbol{\psi}(i\to\cdot) = \boldsymbol{\psi}_{\text{pred}}, \tag{3.4a}$$

$$\boldsymbol{\psi}(i\to j) = \boldsymbol{\psi}_{\text{UA}}, \tag{3.4b}$$

and each label gets a vector:

$$\boldsymbol{\psi}(i \xrightarrow{\ell} j) = \boldsymbol{\psi}_{\text{LA}}(\ell). \tag{3.4c}$$

Scoring. Finally, we use an inner product to score first-order structures:

$$s(p) = \boldsymbol{\phi}(p) \cdot \boldsymbol{\psi}(p). \tag{3.5}$$

Figure 3.2 illustrates our basic model's architecture.

3.1.3 Learning

The parameters of the model are learned using a max-margin objective. Informally, the goal is to learn parameters for the score function so that the gold parse is scored over every incorrect parse with a margin proportional to the cost of the incorrect parse. More formally, let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training set consisting of N pairs of sentence x_i and its gold parse y_i. Training is then the following ℓ_2-regularized empirical risk minimization problem:

$$\min_{\Theta} \frac{\lambda}{2}\|\Theta\|^2 + \frac{1}{N}\sum_{i=1}^{N} L(x_i, y_i; \Theta), \tag{3.6}$$

where Θ is the collection of all parameters in the model, and L is the structured hinge loss:

$$L(x_i, y_i; \Theta) = \max_{y \in \mathcal{Y}(x_i)} \{S(x_i, y) + c(y, y_i)\} - S(x_i, y_i). \tag{3.7}$$

c is a weighted Hamming distance that trades off between precision and recall (Taskar et al., 2004). Following Martins and Almeida (2014), we encourage recall over precision by using the costs 0.6 for false negative arc predictions and 0.4 for false positives.

	Model	DM	PAS	PSD	Avg.
id	Du et al., 2015	89.1	91.3	75.7	86.3
	A&M, 2015	88.2	90.9	76.4	86.0
	BASIC	**89.4**	<u>**92.2**</u>	<u>**77.6**</u>	<u>**87.4**</u>
ood	Du et al., 2015	81.8	87.2	73.3	81.7
	A&M, 2015	81.8	86.9	74.8	82.0
	BASIC	<u>**84.5**</u>	<u>**88.3**</u>	<u>**75.3**</u>	<u>**83.6**</u>

Table 3.1: Labeled parsing performance (F_1 score) on both in-domain (id) and out-of-domain (ood) test data. The last column shows the micro-average over the three tasks. Bold font indicates best performance without syntax. Underlines indicate statistical significance with Bonferroni (1936) correction compared to the best baseline system.[3]

3.1.4 Experiments

We evaluate our basic model on the English dataset from SemEval 2015 Task 18 closed track.[2] We split as in previous work (Almeida and Martins, 2015; Du et al., 2015), resulting in 33,964 training sentences from §00–19 of the WSJ corpus, 1,692 development sentences from §20, 1,410 sentences from §21 as in-domain test data, and 1,849 sentences sampled from the Brown Corpus as out-of-domain test data.

The *closed* track differs from the *open* and *gold* tracks in that it does not allow access to any syntactic analyses. In the open track, additional machine generated syntactic parses are provided, while the gold-track gives access to various gold-standard syntactic analyses. Our model is evaluated with closed track data; it does not have access to any syntactic analyses during training or test.

We refer the readers to §A.1 for implementation details, including training procedures, hyperparameters, pruning techniques, etc..

Empirical results. As our model uses no explicit syntactic information, the most comparable models to ours are two state-of-the-art closed track systems due to Du et al. (2015) and Almeida and Martins (2015). Du et al. (2015) rely on graph-tree transformation techniques proposed by Du

[2]`http://sdp.delph-in.net`
[3]Paired bootstrap, $p < 0.05$ after Bonferroni correction.

et al. (2014), and apply a voting ensemble to well-studied tree-oriented parsers. Closely related to ours is Almeida and Martins (2015), who used rich, hand-engineered second-order features and AD^3 for inference.

Table 3.1 compares our basic model to both baseline systems (labeled F_1 score) on SemEval 2015 Task 18 test data. Scores of those systems are repeated from the official evaluation results. Our basic model significantly outperforms the best published results with a 1.1% absolute improvement on the in-domain test set and 1.6% on the out-of-domain test set.

3.2 Multitask SDP

We introduce two extensions to our single-task model, both of which use training data for all three formalisms to improve performance on each formalism's parsing task. We describe a first-order model, where representation functions are enhanced by parameter sharing while inference is kept separate for each task (§3.2.2). We then introduce a model with cross-task higher-order structures that uses joint inference *across* different tasks (§3.2.3). Both multitask models use AD^3 for decoding, and are trained with the same margin-based objective, as in our single-task model.

3.2.1 Problem Formulation

We will use an additional superscript $t \in \mathcal{T}$ to distinguish the three tasks (e.g., $y^{(t)}$, $\boldsymbol{\phi}^{(t)}$), where $\mathcal{T} = \{\mathrm{DM}, \mathrm{PAS}, \mathrm{PSD}\}$. Our task is now to predict three graphs $\{y^{(t)}\}_{t \in \mathcal{T}}$ for a given input sentence x. Multitask SDP can also be understood as parsing x into a single unified *multigraph* $y = \bigcup_{t \in \mathcal{T}} y^{(t)}$. Similarly to Equations 3.1–3.2, we decompose y's score $S(x, y)$ into a sum of local scores for local structures in y, and we seek a multigraph $\hat{y}$ that maximizes $S(x, y)$.

3.2.2 Multitask SDP with Parameter Sharing

A common approach when using BiLSTMs for multitask learning is to share the BiLSTM part of the model across tasks, while training specialized classifiers for each task (Søgaard and Goldberg, 2016). In this spirit, we let each task keep its own specialized MLPs, and explore two variants of our model that share parameters at the BiLSTM level.

The first variant consists of a set of task-specific BiLSTM encoders as well as a common one that is shared across all tasks. We denote it FREDA. FREDA uses a neural generalization of "frustratingly easy" domain adaptation (Daumé III, 2007; Kim et al., 2016b), where one augments domain-specific features with a shared set of features to capture global patterns. Formally, let $\{\mathbf{h}^{(t)}\}_{t\in\mathcal{T}}$ denote the three task-specific encoders. We introduce another encoder $\widetilde{\mathbf{h}}$ that is shared across all tasks. Then a new set of input functions $\{\boldsymbol{\phi}^{(t)}\}_{t\in\mathcal{T}}$ can be defined as in Equations 3.3a–3.3c, for example:

$$\boldsymbol{\phi}^{(t)}(i \xrightarrow{\ell} j) = \tanh\bigl(\mathbf{C}_{\text{LA}}^{(t)}\bigl[\mathbf{h}_i^{(t)};\mathbf{h}_j^{(t)};\widetilde{\mathbf{h}}_i;\widetilde{\mathbf{h}}_j\bigr] + \mathbf{b}_{\text{LA}}^{(t)}\bigr). \tag{3.8}$$

The predicate and unlabeled arc versions are analogous. The output representations $\{\boldsymbol{\psi}^{(t)}\}$ remain task-specific, and the score is still the inner product between the input representation and the output representation.

The second variant, which we call SHARED, uses *only* the shared encoder $\widetilde{\mathbf{h}}$, and doesn't use task-specific encoders $\{\mathbf{h}^{(t)}\}$. It can be understood as a special case of FREDA where the dimensions of the task-specific encoders are 0.

3.2.3 Multitask SDP with Cross-Task Structures

In syntactic parsing, higher-order structures have commonly been used to model interactions between multiple adjacent arcs in the same dependency tree (*e.g.,* Carreras, 2007; Smith and Eisner, 2008; Martins et al., 2009; Zhang et al., 2014). Lluís et al. (2013), in contrast, used second-order structures to jointly model syntactic dependencies and semantic roles. Similarly, we use higher-order structures *across* tasks instead of *within* tasks. In this work, we look at interactions between arcs that share the same head and modifier.[4] See Figures 3.1b and 3.1c for examples of higher-order cross-task structures.

Higher-order structure scoring. Borrowing from Lei et al. (2014), we introduce a low-rank tensor scoring strategy that, given a higher-order structure p, models interactions between the first-order structures (i.e., arcs) p is made up of. This approach builds on and extends the parameter sharing techniques in §3.2.2. It can either follow FREDA or SHARED to get the input representations for first-order structures.

We first introduce basic tensor notation. The **order** of a tensor is the number of its dimensions. The **outer product** of two vectors forms a second-order tensor (matrix) where $[\mathbf{u} \otimes \mathbf{v}]_{i,j} = u_i v_j$. We denote the **inner product** of two tensors of the same dimensions by $\langle \cdot, \cdot \rangle$, which first takes their element-wise product, then sums all the elements in the resulting tensor.

For example, let p be a labeled third-order structure, including one labeled arc from each of the three different tasks: $p = \{p^{(t)}\}_{t \in \mathcal{T}}$. Intuitively, $s(p)$ should capture every pairwise interaction between the three input and three output representations of p. Formally, we want the score function to include a parameter for each term in the outer product of the representation vectors: $s(p) =$

$$\left\langle \boldsymbol{\mathcal{W}}, \bigotimes_{t \in \mathcal{T}} \left(\boldsymbol{\phi}^{(t)} \left(p^{(t)} \right) \otimes \boldsymbol{\psi}^{(t)} \left(p^{(t)} \right) \right) \right\rangle, \tag{3.9}$$

where $\boldsymbol{\mathcal{W}}$ is a sixth-order tensor of parameters.[5]

With typical dimensions of representation vectors, this leads to an unreasonably large number

[4]In the future we hope to model structures over larger motifs, both across and within tasks, to potentially capture when an arc in one formalism corresponds to a path in another formalism, for example.

[5]This is, of course, not the only way to model interactions between several representations. For instance, one could concatenate them and feed them into another MLP. Our preliminary experiments in this direction suggested that it may be less effective given a similar number of parameters, but we did not run full experiments.

of parameters. Following Lei et al. (2014), we upper-bound the rank of $\mathcal{W}$ by r to limit the number of parameters (r is a hyperparameter, decided empirically). Using the fact that a tensor of rank at most r can be decomposed into a sum of r rank-1 tensors (Hitchcock, 1927), we reparameterize $\mathcal{W}$ to enforce the low-rank constraint by construction:

$$\mathcal{W} = \sum_{j=1}^{r} \bigotimes_{t \in \mathcal{T}} \left(\left[\mathbf{U}_{\text{LA}}^{(t)}\right]_{j,:} \otimes \left[\mathbf{V}_{\text{LA}}^{(t)}\right]_{j,:} \right), \tag{3.10}$$

where $\mathbf{U}_{\text{LA}}^{(t)}, \mathbf{V}_{\text{LA}}^{(t)} \in \mathbb{R}^{r \times d}$ are now our parameters. $[\cdot]_{j,:}$ denotes the jth row of a matrix. Substituting this back into Equation 3.9 and rearranging, the score function $s(p)$ can then be rewritten as:

$$\sum_{j=1}^{r} \prod_{t \in \mathcal{T}} \left[\mathbf{U}_{\text{LA}}^{(t)} \boldsymbol{\phi}^{(t)}\left(p^{(t)}\right)\right]_j \left[\mathbf{V}_{\text{LA}}^{(t)} \boldsymbol{\psi}^{(t)}\left(p^{(t)}\right)\right]_j . \tag{3.11}$$

We refer readers to Kolda and Bader (2009) for mathematical details.

For labeled higher-order structures our parameters consist of the set of six matrices, $\{\mathbf{U}_{\text{LA}}^{(t)}\} \cup \{\mathbf{V}_{\text{LA}}^{(t)}\}$. These parameters are shared between second-order and third-order labeled structures. Labeled *second-order* structures are scored as Equation 3.11, but with the product extending over only the two relevant tasks. Concretely, only four of the representation functions are used rather than all six, along with the four corresponding matrices from $\{\mathbf{U}_{\text{LA}}^{(t)}\} \cup \{\mathbf{V}_{\text{LA}}^{(t)}\}$. *Unlabeled* cross-task structures are scored analogously, reusing the same representations, but with a separate set of parameter matrices $\{\mathbf{U}_{\text{UA}}^{(t)}\} \cup \{\mathbf{V}_{\text{UA}}^{(t)}\}$.

Note that we are not doing tensor factorization; we are learning $\mathbf{U}_{\text{LA}}^{(t)}, \mathbf{V}_{\text{LA}}^{(t)}, \mathbf{U}_{\text{UA}}^{(t)}$, and $\mathbf{V}_{\text{UA}}^{(t)}$ directly, and $\mathcal{W}$ is never explicitly instantiated.

Inference and learning. Given a sentence, we use AD^3 to jointly decode all three formalisms.[6] The training objective used for learning is the sum of the losses for individual tasks. Hyperparam-

[6]Joint inference comes at a cost; our third-order model is able to decode roughly 5.2 sentences (i.e., 15.5 task-specific graphs) per second on a single Xeon E5-2690 2.60GHz CPU.

eters are the same as those in the basic model, and are described in Appendix A.1. One notable difference is that we apply early stopping based on the micro-averaged labeled F_1 score on the development set.

3.2.4 Experiments

Experimental settings. We compare four multitask variants to the basic model, as well as the two baseline systems introduced in §3.1.4.

- SHARED1 is a first-order model. It uses a single shared BiLSTM encoder, and keeps the inference separate for each task.
- FREDA1 is a first-order model based on "frustratingly easy" parameter sharing. It uses a shared encoder as well as task-specific ones. The inference is kept separate for each task.
- SHARED3 is a third-order model. It follows SHARED1 and uses a single shared BiLSTM encoder, but additionally employs cross-task structures and inference.
- FREDA3 is also a third-order model. It combines FREDA1 and SHARED3 by using both "frustratingly easy" parameter sharing and cross-task structures and inference.

In addition, we also examine the effects of syntax by comparing our models to the state-of-the-art open track system (Almeida and Martins, 2015).[7]

Main results overview. Table 3.2 compares our models to the best published results (labeled F_1 score) on SemEval 2015 Task 18. On the in-domain test set, our basic model improves over all closed track entries in all formalisms. It is even with the best open track system for DM and PSD, but improves on PAS and on average, without making use of any syntax. Three of our four multitask variants further improve over our basic model; SHARED1's differences are statistically insignificant. Our best models (SHARED3, FREDA3) outperform the previous state-of-the-art closed track system

[7]Kanerva et al. (2015) was the winner of the gold track, which overall saw higher performance than the closed and open tracks. Since gold-standard syntactic analyses are not available in most realistic scenarios, we do not include it in this comparison.

	In-domain				Out-of-domain			
	DM	**PAS**	**PSD**	**Avg.**	**DM**	**PAS**	**PSD**	**Avg.**
Du et al., 2015	89.1	91.3	75.7	86.3	81.8	87.2	73.3	81.7
A&M, 2015 (closed)	88.2	90.9	76.4	86.0	81.8	86.9	74.8	82.0
A&M, 2015 (open)†	89.4	91.7	77.6	87.1	83.8	87.6	76.2	83.3
BASIC	89.4	<u>92.2</u>	77.6	87.4	<u>84.5</u>	<u>88.3</u>	75.3	83.6
SHARED1	89.7	91.9	77.8	87.4	<u>84.4</u>	<u>88.1</u>	75.4	83.5
FREDA1	<u>90.0</u>	<u>92.3</u>	<u>78.1</u>	<u>87.7</u>	<u>84.9</u>	<u>88.3</u>	75.8	<u>83.9</u>
SHARED3	<u>90.3</u>	<u>92.5</u>	<u>**78.5**</u>	<u>**88.0**</u>	<u>**85.3**</u>	<u>88.4</u>	76.1	<u>84.1</u>
FREDA3	<u>**90.4**</u>	<u>**92.7**</u>	<u>**78.5**</u>	<u>**88.0**</u>	<u>**85.3**</u>	<u>**89.0**</u>	**76.4**	<u>**84.4**</u>

Table 3.2: Labeled F_1 score on the in-domain and out-of-domain test sets. The last columns show the micro-average over the three tasks. † denotes the use of syntactic parses. Bold font indicates best performance among all systems, and underlines indicate statistical significance with Bonferroni correction compared to A&M, 2015 (open), the strongest baseline system.

by 1.7% absolute F_1, and the best open track system by 0.9%, without the use of syntax.

We observe similar trends on the out-of-domain test set, with the exception that, on PSD, our best-performing model's improvement over the open-track system of Almeida and Martins (2015) is not statistically significant.

The extent to which we might benefit from syntactic information remains unclear. With automatically generated syntactic parses, Almeida and Martins (2015) manage to obtain more than 1% absolute improvements over their closed track entry, which is consistent with the extensive evaluation by Zhang et al. (2016), but we leave the incorporation of syntactic trees to future work. Syntactic parsing could be treated as yet another output task, as explored in Lluís et al. (2013) and in the transition-based frameworks of Henderson et al. (2013) and Swayamdipta et al. (2016).

Effects of structural overlap. We hypothesized that the overlap between formalisms would enable multitask learning to be effective; in this section we investigate in more detail how structural overlap affected performance. By looking at *undirected* overlap between unlabeled arcs, we discover

	Undirected			Directed		
	DM	**PAS**	**PSD**	**DM**	**PAS**	**PSD**
DM	-	67.2	56.8	-	64.2	26.1
PAS	70.0	-	54.9	66.9	-	26.1
PSD	57.4	56.3	-	26.4	29.6	-

Table 3.3: Pairwise structural similarities between the three formalisms in unlabeled F_1 score. Scores from Oepen et al. (2015).

that modeling only arcs in the same direction may have been a design mistake.

DM and PAS are more structurally similar to each other than either is to PSD. Table 3.3 compares the structural similarities between the three formalisms in unlabeled F_1 score (each formalism's gold-standard unlabeled graph is used as a prediction of each other formalism's gold-standard unlabeled graph). All three formalisms have more than 50% overlap when ignoring arcs' directions, but considering direction, PSD is clearly different; PSD reverses the direction about half of the time it shares an edge with another formalism. A concrete example can be found in Figure 2.1, where DM and PAS both have an arc from *"Last"* to *"week,"* while PSD has an arc from *"week"* to *"Last."*

We can compare FREDA3 to FREDA1 to isolate the effect of modeling higher-order structures. Table 3.4 shows performance on the development data in both unlabeled and labeled F_1. We can see that FREDA3's unlabeled performance improves on DM and PAS, but *degrades* on PSD. This supports our hypothesis, and suggests that in future work, a more careful selection of structures to model might lead to further improvements.

3.3 Related Work

We note two important strands of related work.

	DM		PAS		PSD	
	UF	LF	UF	LF	UF	LF
FREDA1	91.7	90.4	93.1	91.6	89.0	79.8
FREDA3	91.9	90.8	93.4	92.0	88.6	80.4

Table 3.4: Unlabeled (UF) and labeled (LF) parsing performance of FREDA1 and FREDA3 on the development set of SemEval 2015 Task 18.

Graph-based parsing. Graph-based parsing was originally invented to handle non-projective syntax (*e.g.,* McDonald et al., 2005; Koo et al., 2010; Martins et al., 2013), but has been adapted to semantic parsing (*e.g.,* Flanigan et al., 2014; Martins and Almeida, 2014; Thomson et al., 2014; Kuhlmann, 2014). Local structure scoring was traditionally done with linear models over hand-engineered features, but lately, various forms of representation learning have been explored to learn feature combinations (*e.g.,* Lei et al., 2014; Taub-Tabib et al., 2015; Pei et al., 2015). Our work is perhaps closest to those who used BiLSTMs to encode inputs (Kiperwasser and Goldberg, 2016; Kuncoro et al., 2016; Wang and Chang, 2016; Dozat and Manning, 2017; Ma and Hovy, 2016).

Multitask learning in NLP. There have been many efforts in NLP to use joint learning to replace pipelines, motivated by concerns about cascading errors. Collobert and Weston (2008) proposed sharing the same word representation while solving multiple NLP tasks. Zhang and Weiss (2016) use a continuous stacking model for POS tagging and parsing. Ammar et al. (2016) and Guo et al. (2016) explored parameter sharing for multilingual parsing. Johansson (2013) and Kshirsagar et al. (2015) applied ideas from domain adaptation to multitask learning. Successes in multitask learning have been enabled by advances in representation learning as well as earlier explorations of parameter sharing (Ando and Zhang, 2005; Blitzer et al., 2006; Daumé III, 2007).

3.4 Conclusion

We showed two orthogonal ways to apply deep multitask learning to graph-based parsing. The first shares parameters when encoding tokens in the input with recurrent neural networks, and the second introduces interactions between output structures across formalisms. Without using syntactic parsing, these approaches outperform even state-of-the-art semantic dependency parsing systems that use syntax. Because our techniques apply to labeled directed graphs in general, they can easily be extended to incorporate more formalisms, semantic or otherwise. In future work we hope to explore cross-task scoring and inference for tasks where parallel annotations are not available. Our code is open-source and available at `https://github.com/Noahs-ARK/NeurboParser`.

Chapter 4

Graph-based Semantic Parsing as a Steiner Problem

[*] Our next contribution is a theoretical one, with potential applications in semantic graph prediction. We show that decoding graphs under a *connected* constraint but without a *spanning* constraint results in an NP-hard problem, by reduction from the prize-collecting Steiner tree (PCST) problem (Goemans and Williamson, 1992). We introduce the **node-and-edge-weighted connected subgraph (NEWCS)** problem, an extension of PCST where both node and edge weights may be positive or negative. The NEWCS problem naturally arises when jointly predicting the nodes and edges of a semantic graph. In particular, formalisms such as the **Abstract Meaning Representation** (**AMR**; Banarescu et al., 2013; Dorr et al., 1998) and **Minimal Recursion Semantics** (**MRS**; Copestake et al., 2005) include abstract concept nodes that do not correspond directly to tokens in the sentence. Parsing to these representations requires determining the set of labeled nodes in the graph, a subtask dubbed **concept identification**. Concept identification for these formalisms is difficult (*e.g.,* Wang et al., 2015; Flanigan et al., 2016; Buys and Blunsom, 2017), which makes joint modeling of

[*] *The work described in this chapter was done jointly with Jeffrey Flanigan, Benjamin J. Plaut, and Noah A. Smith.*

concepts and edges an enticing direction to explore.[1]

We adapt two known techniques from PCST solving to NEWCS solving. First, we show that the integer linear program that we previously used as a PCST solver (Liu et al., 2015) can be extended to NEWCS solving (§4.3). Second, we combine our **maximum spanning connected graph (MSCG)** algorithm (Flanigan et al., 2014) with a relaxation due to Beasley (1989) to produce a novel approximate NEWCS solver (§4.4). We also introduce a novel set of *tests* which can transform a NEWCS instance into an instance with fewer nodes (§4.5). An optimal solution to the original problem instance can then be recovered from a solution for the reduced instance. Finally, we note the connection of this work to other applications in natural language processing (§4.6).

4.1 Background

4.1.1 The Maximum Spanning Connected Graph Algorithm

In Flanigan et al. (2014) we introduced the maximum spanning connected graph algorithm (MSCG). We don't claim MSCG as a contribution of this thesis, but MSCG forms a core component of our novel NEWCS algorithm (§4.4), so we review it here. The input to MSCG is an undirected graph $G = (V_G, E_G)$ with real-valued edge weights $w(e) \in \mathbb{R}, \forall e \in E_G$, and it returns a spanning connected subgraph $H \subseteq G$ of maximal weight:

$$\text{MSCG}(G, w) = \arg\max_{\substack{H \subseteq G, \\ H \text{ spanning}, \\ H \text{ connected}}} \sum_{e \in E_H} w(e). \tag{4.1}$$

Only edge weights are considered, as spanning subgraphs must include every node.

The algorithm extends Kruskal's algorithm for finding minimum spanning trees (Kruskal, 1956). Whereas Kruskal's algorithm assumes that every edge weight is negative, MSCG handles

[1]Note that in semantic dependency parsing, concept identification is the same as singleton classification (§2.2.1). Since classifying singletons can already be done with very high accuracy, we expect the findings in this chapter to be more helpful for AMR and MRS than for SDP.

nonnegative edges as well, and as a result may not output a tree. MSCG consists of two steps:

1. Add all edges with nonnegative weight in E_G to E_H.

2. Pick the least negative edge in E_G that has not been considered yet. If it connects two disconnected components in E_H, add it to E_H. Otherwise discard it.

Step 2 is repeated until the graph is connected. Flanigan et al. (2014) contains a proof that this greedy algorithm gives the optimal solution. It requires sorting edges by weight, so its runtime is $O(|E_G| \log |E_G|)$, which is $O(n^2 \log n)$ when G is dense.

4.1.2 The Prize-collecting Steiner Tree Problem

First let us formally state the prize-collecting Steiner tree problem (Goemans and Williamson, 1992; Bienstock et al., 1993). Let $G = (V_G, E_G)$ be an undirected graph with positive node weights, $w(v) \geq 0, \forall v \in V_G$, and negative edge weights, $w(e) \leq 0, \forall e \in E_G$. The **prize-collecting Steiner tree (PCST)** problem is to find a connected (but not necessarily spanning) subgraph $H = (V_H, E_H) \subseteq G$ of maximal weight:

$$\text{PCST}(G, w) = \arg\max_{\substack{H \subseteq G, \\ H \text{ connected}}} \left\{ \sum_{v \in V_H} w(v) + \sum_{e \in E_H} w(e) \right\}. \tag{4.2}$$

The **node-weighted Steiner tree (NWST)** problem is a similar variant which allows positive or negative node weights (Segev, 1987). In both cases though, edge weights are required to be negative, so the solution is guaranteed to be a tree.

Steiner problems are of interest to utility companies, for instance, who might wish to connect paying customers to their grid in such a way as to maximize profits. PCST is an extension of the original Steiner tree problem (in which every node has weight zero or infinity), and both are NP-complete (Karp, 1972). Hundreds of papers have been written on variants of Steiner tree problems; see (Hwang et al., 1992) for a survey. Remarkably, to the best of our knowledge, no work has been done on the problem with unrestricted edge and node weights, and most techniques for

Steiner problems crucially rely on the guarantee that the solution will be a tree. While constraining edge weights to be negative makes sense as a model of the cost of running electricity lines, and constraining node weights to be positive makes sense as a model of the expected value of potential customers, neither constraint is appropriate when predicting semantic graphs. Positive edge weights are necessary in order to predict reentrancies, which semantic graphs are known to have. And negative node weights are needed to give a model the power to express that a concept is more likely *not* to appear in a parse. In the next section we introduce an extension of PCST that removes these constraints, and is more suitable for semantic parsing.

4.2 The Node-and-Edge-weighted Connected Subgraph (NEWCS) Problem

Here we define a novel extension of the PCST problem in which node weights are not required to be positive and edge weights are not required to be negative. Let $G = (V_G, E_G)$ be an undirected graph with real-valued node weights, $w(v) \in \mathbb{R}, \forall v \in V_G$, and real-valued edge weights, $w(e) \in \mathbb{R}, \forall e \in E_G$. The **node-and-edge-weighted connected subgraph (NEWCS)** problem is to find a connected, but not necessarily spanning subgraph $H = (V_H, E_H) \subseteq G$ of maximal weight:

$$\text{NEWCS}(G, w) = \arg\max_{H \subseteq G, \atop H \text{ connected}} \left\{ \sum_{v \in V_H} w(v) + \sum_{e \in E_H} w(e) \right\}. \tag{4.3}$$

In the remainder of the chapter, we will assume $G = (V_G, E_G)$ is the input graph, and $H = (V_H, E_H) \subseteq G$ is a connected subgraph of G of maximal weight. We will refer to the number of nodes in G as n.[2] Equation 4.3 is exactly the same as Equation 4.2; we have only removed the conditions on the weight function w. This allows us to state and prove the following trivial theorem:

[2]For dependency parsing, n coincides with the number of tokens in the sentence, but for joint semantic parsing n is usually larger.

Theorem 1. *The* NEWCS *problem is NP-hard.*

Proof. NEWCS is a relaxation of PCST, so every PCST problem is a NEWCS problem. Since PCST can be reduced to NEWCS, NEWCS is also NP-hard. □

We can situate this with respect to other results in graph-based parsing to give a more complete picture of available techniques and their complexities. MSTParser (McDonald et al., 2005) was originally a first-order non-projective dependency parsing model (i.e. each arc is scored independently) with $O(n^2)$ inference via the Chu-Liu-Edmonds algorithm (CLE; Chu and Liu, 1965; Edmonds, 1967). Second-order MST parsing (i.e. motifs of two adjacent arcs can be scored together) was subsequently shown to be NP-complete (McDonald and Pereira, 2006). This led researchers to explore approximate inference techniques for fast higher-order non-projective parsing (*e.g.,* Koo et al., 2010; Martins et al., 2013). JAMR (Flanigan et al., 2014) extended MST parsing techniques from arborescences to (non-tree) semantic graphs, with an arc-factored model and $O(n^2 \log n)$ inference via MSCG. Theorem 1 shows that the spanning constraint was computationally critical; removing it results in an NP-hard problem, even without higher-order factors. This result, along with preliminary experiments, leads us to believe that approximate inference is required if graph-based methods are to be practical for joint node and edge prediction. We propose one such method in §4.4. First, as a reasonable baseline, we give a formulation of the problem as an integer linear program (ILP), which can be solved either exactly or approximately by off-the-shelf ILP solvers like Gurobi[3] or CPLEX.[4]

4.3 An Integer Linear Program Formulation

In Liu et al. (2015) we gave an ILP formulation of the PCST problem, which was adapted and modified from two similar formulations (Ljubić et al., 2006; Martins et al., 2009). We used the

[3]http://www.gurobi.com/

[4]https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer

solver as part of a summarization system that condensed an AMR representation of a document into a smaller "summary" AMR graph. In that work, we enforced a tree constraint, but the tree constraint could easily be removed, resulting a NEWCS formulation, which we give here. The formulation converts the undirected input graph into a directed one, and uses **single commodity flow** variables to enforce connectivity. The direction of edges in the solution are then ignored to recover a maximal undirected subgraph.

More formally, we convert each undirected edge $e = u{\leftrightarrow}v \in E_G$ into two directed arcs $u{\rightarrow}v$, and $v{\rightarrow}u$, both with weight $w(e)$. Let A_G be the set of all such directed arcs from the input graph, and let A_H be the set of arcs in the directed subgraph output by the ILP. Let $\mathbb{1}$ be the indicator function:

$$\mathbb{1}(P) = \begin{cases} 1 & \text{if } P \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

For each node $v \in V_G$ in the input graph, we have a binary variable that indicates whether v is in the solution:

$$z_v = \mathbb{1}(v \in V_H). \tag{4.5}$$

Similarly, each arc gets a binary variable:

$$z_{u\rightarrow v} = \mathbb{1}(u{\rightarrow}v \in A_H). \tag{4.6}$$

To ensure that we can recover a valid undirected graph from the solution, we constrain that at most one direction is included:

$$z_{u\rightarrow v} + z_{v\rightarrow u} \leq 1, \forall u{\leftrightarrow}v \in E_G. \tag{4.7}$$

And for every arc $u{\rightarrow}v \in A_G$, we constrain that it may only be included if both its endpoints u and

v are included:

$$z_{u\to v} \leq z_u, \forall u\to v \in A_G, \tag{4.8a}$$

$$z_{u\to v} \leq z_v, \forall u\to v \in A_G. \tag{4.8b}$$

We enforce connectivity using a set of single commodity flow variables, f_a, one for each arc $a \in A_G$. Each flow variable takes a nonnegative integer value, representing the flow over a. We introduce an extra node ROOT, and an arc from ROOT to each other node $v \in V_G$. We constrain that at most one of these edges can be in the solution:

$$\sum_{v\in V_G} z_{\text{ROOT}\to v} \leq 1. \tag{4.9}$$

For each directed arc $a \in A_G$, we introduce a flow variable f_a. ROOT may send out up to n units of flow (one unit for each node in the solution):

$$\sum_{v\in V_G} f_{\text{ROOT}\to v} \leq n. \tag{4.10}$$

Flow may only be sent over an arc if the arc is included in the solution:

$$0 \leq f_a \leq n z_a, \forall a \in A_G. \tag{4.11}$$

Each non-root node v must "consume" exactly one unit of flow if it is included in the solution, and zero units otherwise. By "consume", we mean that the flow over its outgoing arcs is one unit less than the flow over its incoming arcs:

$$\sum_{s\in V_G \bigcup\{\text{ROOT}\}} f_{s\to v} - \sum_{t\in V_G} f_{v\to t} = z_v, \forall v \in V_G. \tag{4.12}$$

Together, the flow variables and flow constraints ensure that every node in the solution is reachable from ROOT. Since ROOT can only have one outgoing arc (by Equation 4.9), the solution consists of at most one connected component. The problem may now be solved by an ILP solver, by maximizing the following linear objective:

$$\arg\max \mathbf{z}, \mathbf{f} \left\{ \sum_{v \in V_G} w(v) z_v + \sum_{u \to v \in A_G} w(u \leftrightarrow v) z_{u \to v} \right\}, \tag{4.13}$$

subject to all of the given linear constraints (Equations 4.7–4.12).[5]

Symmetry Breaking

The ILP as formulated contains spurious ambiguity—multiple solutions to the ILP correspond to the exact same undirected subgraph. One source of ambiguity is that given an optimal solution, any one of its nodes may be selected as the one directly attached to ROOT. Ljubić et al. (2006) introduced a set of *symmetry-breaking* constraints which disallow all but one solution. They found that even though symmetry breaking adds $O(n^2)$ constraints, CPLEX returned a solution faster. In our experiments using Gurobi, we found the opposite. We note that in semantic parsing, there is a more natural way to break the symmetry. It is to jointly predict the "top" (§2.2.3) of the graph, and incorporate each node v's "top" score into the weight of ROOT$\to v$.

4.4 An Approximate Algorithm

In preliminary AMR parsing experiments, we found ILP solving using Gurobi to be prohibitively slow, with exact solutions sometimes taking hours, and even approximate solutions taking multiple minutes. Inspired by Beasley's Steiner tree solving algorithm (Beasley, 1989), along with similar

[5]Note that the objective is linear with respect to $\mathbf{z}$ regardless of the form or parameterization of the scoring function w. In other words, w need not be the output of a linear model; it could be the output of a neural network, or any other function. We precompute it before decoding, and during decoding consider it to be a single fixed scalar per candidate node or arc.

successful approaches in syntactic dependency parsing (Koo et al., 2010; Rush et al., 2012; Martins et al., 2013), we investigated combining a discrete graph algorithm with Lagrangian relaxation. Here we introduce a novel approximate algorithm, **BEASLEYGRAPH**, which extends Beasley's algorithm to handle NEWCS problems (and hence PCST problems as well).

At a high level, Beasley's algorithm modifies the input graph and adds constraints in order to turn the problem of finding a non-spanning tree into one of finding a spanning tree subject to constraints. In BEASLEYGRAPH, instead of finding a maximal spanning tree, we find a maximal spanning connected subgraph. As noted in §4.2, finding spanning connected subgraphs is computationally easier than finding (possibly non-spanning) connected subgraphs. The constraints we add ensure that an optimal NEWCS solution can be recovered from the solution to the connected spanning subgraph problem. The constraints can be approximately enforced with Lagrangian relaxation (Geoffrion, 1974; Fisher, 2004), or more specialized methods like AD^3 (Martins et al., 2011), to get an approximate NEWCS algorithm.

Formally, let $G = (V_G, E_G)$ and $w : (V_G \cup E_G) \to \mathbb{R}$ be a NEWCS problem instance. We modify G by introducing a ROOT node: $V_{G'} = V_G \cup \{\text{ROOT}\}$. We also introduce two labeled edges between ROOT and every other node v, one with label TOP and weight 0,[6] and one with label OMIT and weight $-w(v)$: $E_{G'} = E_G \cup \bigcup_{v \in V_G} \{\text{ROOT} \overset{\text{TOP}}{\longleftrightarrow} v, \text{ROOT} \overset{\text{OMIT}}{\longleftrightarrow} v\}$. Intuitively, an OMIT edge will indicate that its adjacent node does *not* appear in the solution.

We will use a binary indicator variable z_e for each edge e, as before, to express constraints. We constrain that at most one TOP edge can be chosen:

$$\sum_{v \in V_G} z_{\text{ROOT} \overset{\text{TOP}}{\longleftrightarrow} v} \leq 1. \tag{4.14}$$

[6]Similarly to §4.3, when using this solver as part of a semantic parser, a "top" model score could be used as a weight here. This jointly solves "top" prediction along with concept and edge prediction, with the added benefit of breaking symmetry.

And for a given node v, we require that its OMIT edge and its TOP edge cannot both be included:

$$z_{\text{ROOT}\overset{\text{OMIT}}{\longleftrightarrow}v} + z_{\text{ROOT}\overset{\text{TOP}}{\longleftrightarrow}v} \leq 1, \forall v \in V_G. \tag{4.15}$$

Lastly, if a node v's OMIT edge is included, then no other edge adjacent to v may be included:

$$z_{\text{ROOT}\overset{\text{OMIT}}{\longleftrightarrow}v} + z_{v\leftrightarrow u} \leq 1, \forall u, v \in V_G. \tag{4.16}$$

All of these constraints (Equations 4.14–4.16) can be organized into a single system of linear inequalities:

$$A\mathbf{z} \leq \mathbf{1}. \tag{4.17}$$

Now if we can find a maximum spanning connected subgraph $H' \subseteq G'$ satisfying Equation 4.17, we can recover a solution $H \subseteq H'$ to the NEWCS problem. Call a node v **omitted** in H' if $\text{ROOT} \overset{\text{OMIT}}{\longleftrightarrow} v \in E_{H'}$. Likewise, call a node v a **top** in H' if $\text{ROOT} \overset{\text{TOP}}{\longleftrightarrow} v \in E_{H'}$. Let $V_H = \{v \in V_G \mid v \text{ not omitted in } H'\}$ and $E_H = E_{H'} \cap E_G$. In other words, to get H we discard the ROOT node, discard the OMIT and TOP edges, discard all omitted nodes from H', and keep everything else.

Lemma 1. *H is connected.*

Proof. If V_H is empty, then the lemma is vacuously true.

When V_H is nonempty, we will show every node in V_H is connected to a single top node $t \in V_H$, and hence to each other: Let $v \in V_H$ be any non-omitted node. H' is connected and spanning, so v must have a simple path in H' to ROOT. The path p from v to ROOT cannot go through an OMIT edge, because omitted nodes have degree 1 (by Equation 4.16). The path therefore must go through a TOP edge, as ROOT has no other kind of adjacent edges. There is at most one TOP edge (by Equation 4.14); let t be the top node in H'. So p must consist of a path from v to t, and then a TOP edge from t to ROOT. Since p is simple, the path from v to t does not use the TOP edge, so it must use only edges from E_G. Thus there is a path in H from v to t. $t \in V_H$ (by Equation 4.15). Thus every node in H is connected to each other (through t). □

Lemma 2. *The node-and-edge-weight of H is equal to the edge-weight of H' plus a constant:*

$$w(H) = w(H') + C, \tag{4.18}$$

where,

$$w(H) = \sum_{v \in V_H} w(v) + \sum_{e \in E_H} w(e), \tag{4.19}$$

$$w(H') = \sum_{e \in E_{H'}} w(e), \tag{4.20}$$

$$C = \sum_{v \in V_G} w(v). \tag{4.21}$$

Proof. Recall that TOP edges contribute no weight; OMIT edges have the negated weight of their adjacent node; and all original edges keep their original weight. So,

$$w(H') + C = \sum_{e \in E_H} w(e) + \sum_{\text{ROOT} \overset{\text{OMIT}}{\longleftrightarrow} v \in E_{H'}} w(\text{ROOT} \overset{\text{OMIT}}{\longleftrightarrow} v) + C \tag{4.22}$$

$$= \sum_{e \in E_H} w(e) + \sum_{v \text{ omitted in } H'} -w(v) + C \tag{4.23}$$

$$= \sum_{e \in E_H} w(e) + \sum_{v \text{ not omitted in } H'} w(v) \tag{4.24}$$

$$= w(H) \tag{4.25}$$

Note that C is a sum over all nodes in the input graph, and so is constant with respect to H'. □

Lemma 3. *Every connected subgraph $H \subseteq G$ has a corresponding feasible spanning connected subgraph $H' \subseteq G'$.*

Proof. Proof by construction: If H is empty, let $E_{H'} = \{\text{ROOT} \xleftrightarrow{\text{OMIT}} v \mid v \in V_G\}$. Otherwise, arbitrarily select $t \in V_H$. Let $E_{H'} = V_H \cup \{\text{ROOT} \xleftrightarrow{\text{TOP}} t\} \cup \{\text{ROOT} \xleftrightarrow{\text{OMIT}} v \mid v \notin V_H\}$. □

Theorem 2. *H is a solution to the* NEWCS *problem.*

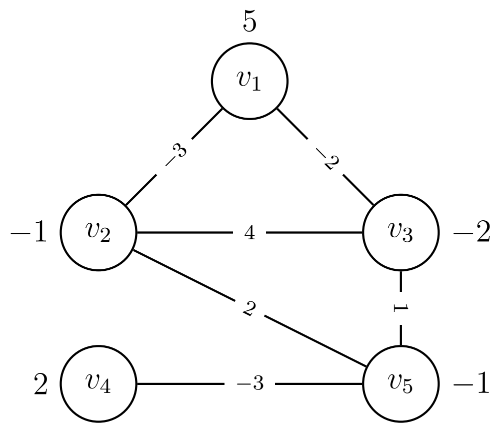
Proof. Lemmas 1 and 3 together show that there is an invertible transformation from connected subgraphs of G to spanning connected subgraphs of G' satisfying Equation 4.17. Lemma 2 shows that their objective values are equal up to a constant, so the maximizer of one is also the maximizer of the other. □

Theorem 2 shows that when the constraints are exactly enforced, we recover an exact solution. When a Lagrangian method is used, sometimes the constraints can be enforced exactly, in which case we can obtain a **certificate of optimality**, but sometimes the method fails to converge. In that case, we resort to a heuristic in order to return a feasible but not necessarily optimal solution. At each iteration during Lagrangian relaxation, MSCG is run on G' with weights that have been adjusted by Lagrange multipliers, outputting a graph H'. Before convergence, H' may not satisfy Equation 4.17. But a key property of Lagrangian relaxation is that the Lagrange-adjusted objective gives an upper bound on the true objective, $w(\text{NEWCS}(G, w))$. We also use H' to find a feasible solution. As our heuristic, we use the highest positive-scoring connected component of H (or the empty graph if all connected components have negative weight). Note that every feasible solution gives a lower bound on the objective. These upper and lower bounds could potentially be used to improve an exact solver in a branch-and-bound framework. In our case though, we simply return the highest-scoring feasible solution over all iterations, along with the lowest upper bound. BEASLEYGRAPH is summarized in Algorithm 1. See Figure 4.1 for an illustration of a successful run of the algorithm.

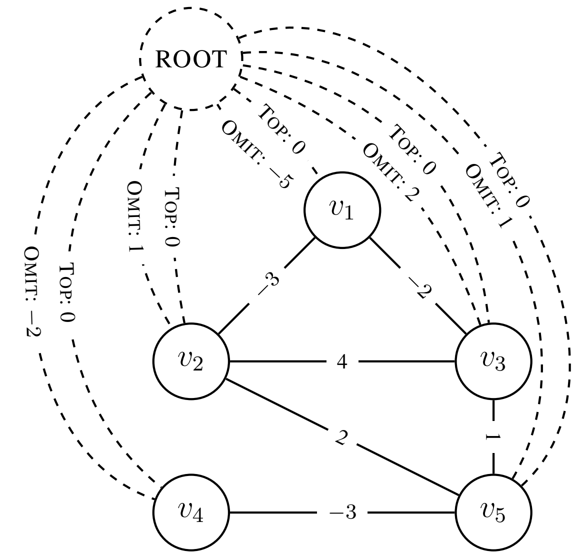
BEASLEYGRAPH is particularly well suited to graph-based semantic parsing, because additional linguistically motivated constraints can be easily added the set of constraints enforced by Lagrangian relaxation. In fact, inference in JAMR (Flanigan et al., 2014, 2016) is already done via MSCG

Algorithm 1: BEASLEYGRAPH

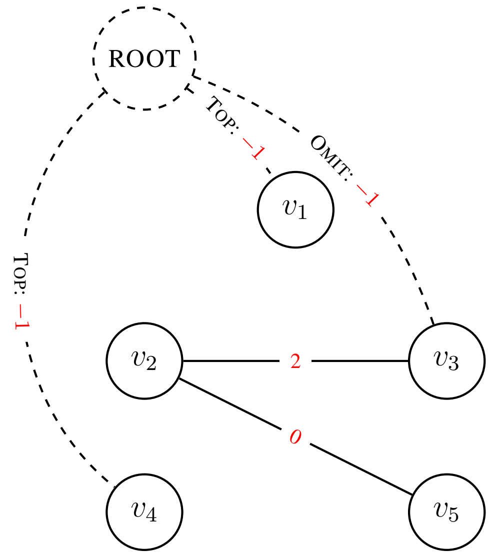
```
Data: A graph $G = (V_G, E_G)$ with node and edge weights given by $w$.
      $K \in \mathbb{N}$, a maximum number of iterations.
      $\eta \in \mathbb{R}^{\geq 0}$, a learning rate.
Result: A connected subgraph of $G$, and an upper bound on the weight of any such
        subgraph.
1  $V_{G'} \leftarrow V_G \cup \{\textsc{root}\}$ ;
2  $E_{G'} \leftarrow E_G \cup \bigcup_{v \in V_G} \{\textsc{root} \overset{\textsc{Top}}{\longleftrightarrow} v, \textsc{root} \overset{\textsc{Omit}}{\longleftrightarrow} v\}$ ;
3  Let $w(\textsc{root} \overset{\textsc{Top}}{\longleftrightarrow} v) = 0$ and $w(\textsc{root} \overset{\textsc{Omit}}{\longleftrightarrow} v) = -w(v)$ for all $v \in V_G$. Let $\mathbf{w}$ be a
     column vector containing all edge weights.
4  Let $\mathcal{L}(\mathbf{z}, \lambda) = \mathbf{w}^\top \mathbf{z} + \lambda^\top(\mathbf{1} - A\mathbf{z})$ ;          // Lagrangian relaxation
5  $\lambda \leftarrow \mathbf{0}$ ;
6  $H^{\text{best}} \leftarrow \emptyset$ ;                                  // best feasible solution
7  $w^{\text{UB}} \leftarrow \sum_{e \in E_{G'}} \max(0, w(e))$ ;                  // lowest upper bound
8  for $i$ in $1, \ldots, K$ do
9      $\mathbf{w}' \leftarrow \mathbf{w} - A^\top \lambda$ ;                     // Lagrange-adjusted weights
10     $H' \leftarrow \text{MSCG}(G', w')$ ; $\mathbf{z} \leftarrow$ binary indicators of $E_{H'}$ ;
11     $H \leftarrow$ highest-weighted connected component of $H' \setminus \{\textsc{root}\}$ ;
12     if $w(H) \geq w(H^{best})$ then
13         $H^{\text{best}} \leftarrow H$ ;
14     if $\mathcal{L}(\mathbf{z}, \lambda) \leq w^{UB}$ then
15         $w^{\text{UB}} \leftarrow \mathcal{L}(\mathbf{z}, \lambda)$ ;
16     if $w(H^{best}) = w^{UB}$ then
17         return $H^{best}$, $w^{UB}$ ;                         // $H^{\text{best}}$ is optimal.
       /* Take a projected gradient step toward minimizing $\mathcal{L}$.
          */
18     $\lambda \leftarrow \lambda - \eta \nabla \mathcal{L}(\mathbf{z}, \lambda)$ ;
19     $\lambda \leftarrow \max(\mathbf{0}, \lambda)$ ;
20 return $H^{best}$, $w^{UB}$ ;                         // $H^{\text{best}}$ is non-optimal.
```



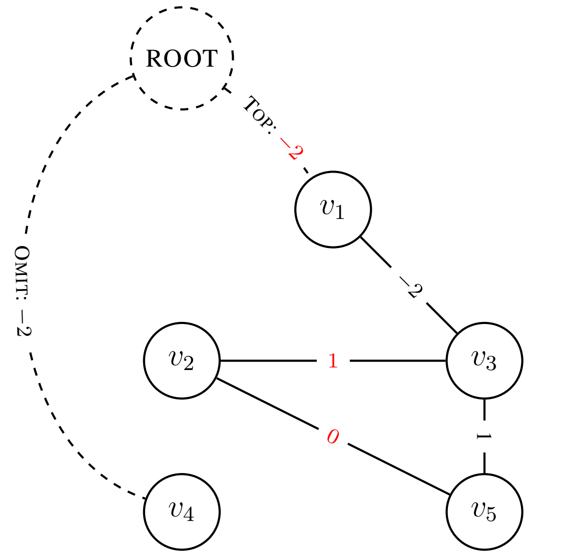
(a) An instance G of the NEWCS problem.



(b) G converted into G', an instance of the MSCG problem. Dashed lines indicate synthetic nodes and edges added during the conversion.



(c) The output of MSCG after an intermediate iteration of Lagrangian relaxation. Weights have been adjusted but H' still does not yet satisfy the constraints.



(d) The output of the final iteration of Lagrangian relaxation. The connected component containing nodes v_1, v_2, v_3, and v_5 is returned.

Figure 4.1: An illustration of a succesful run of BEASLEYGRAPH.

plus Lagrangian relaxation, so the extension to joint concept and relation identification using BEASLEYGRAPH is straight-forward. In the following section, we introduce methods that work for NEWCS (and hence PCST) solving, and so are of interest to the algorithms community. Note though that they may not interact well with additional task-specific constraints, and so may require more care if they are to be used in NLP.

4.5 Reducing the Problem Size with Tests

As with all NP-hard problems, the runtime complexity of NEWCS solving is highly dependent on the size of the input. Memory can be an issue also; for very large PCST instances, ILP solvers will often run out of available memory before returning a solution (Ljubić et al., 2006). A common approach in PCST solving is to run a set of heuristic reduction **tests** to reduce the size of the input graph before sending it to a solver (Duin and Volgenant, 1989). A reduction *test* is a solution-preserving invertible transformation of the input graph: an optimal solution for the transformed graph can be "untransformed" to give an optimal solution for the original graph. Whereas BEASLEYGRAPH transformed the problem into a different problem, tests transform a problem instance into a smaller instance of the same problem. Because of this, tests can be composed and run in sequence; in practice they are tried repeatedly until all tests fail to reduce the problem size.

Our main contribution is a set of tests based on **Steinerpaths** (Duin and Volgenant, 1987). We expand the scope and applicability of Steinerpaths, and propose practical methods for finding them (§4.5.1). We also catalog some tests from prior work on PCST solving, noting which can be easily adapted to NEWCS solving (§**??**).

4.5.1 Steinerpaths

(Duin and Volgenant, 1987) introduced the useful idea of a **Steinerpath**. We present here a simpler, slightly generalized definition, using our own notation and without any assumptions on the sign of weights. Given an undirected input graph $G = (V_G, E_G)$ and node and edge weights $w : (V_G \cup E_G) \to \mathbb{R}$, first let us construct a new directed graph $G^* = (V_G, A_{G^*})$, where

$$A_{G^*} = \bigcup_{u \leftrightarrow v \in E_G} \{u \to v, v \to u\}, \tag{4.26}$$

. G^* has an arc going in both directions for each original edge. Define the weight of an arc to be the weight of its destination plus the weight of its original undirected edge:

$$w(u \to v) = w(v) + w(u \leftrightarrow v). \tag{4.27}$$

These weights can be thought of as the value of adding an arc when its source is already part of the solution but its destination is not. A **Steinerpath** is a simple path $p = \langle v_o \to v_1, v_1 \to v_2, \ldots, v_{m-1} \to v_m \rangle$ in G^* such that every **terminal subpath** (or **suffix**) of p has nonnegative weight:

$$\forall \ell \in \{0, \ldots, m-1\} : \sum_{i=\ell}^{m-1} w(v_i \to v_{i+1}) \geq 0. \tag{4.28}$$

A terminal subpath of p is a path with the same endpoint as p that uses only arcs from p. It follows immediately from the definition that every suffix of a Steinerpath is also a Steinerpath. The following lemma captures why Steinerpaths are of interest.

Lemma 4. *If there is a Steinerpath from v_0 to v_m in G^*, and there is an optimal* NEWCS *solution containing v_0, then there is an optimal solution containing both v_0 and v_m.*

Proof. Lemma 4 is proven for PCST in (Duin and Volgenant, 1987) under a slightly different definition of Steinerpath. Our definition corresponds to a *chain* of Steinerpaths under their original definition. The proof under our definition follows from theirs, and the fact that implication is transitive.

The intuition behind the proof is that if an optimal solution H already contains some non-empty subset of nodes in the path $\{v_{i_1}, \ldots, v_{i_\ell}\}$, with $i_1 < \ldots < i_\ell$, then H could be improved by adding the remainder of the Steinerpath $\langle v_{i_\ell} \to v_{i_\ell+1}, \ldots, v_{m-1} \to v_m \rangle$. $\square$

Another way to state Lemma 4 in terms of indicator variables is that, if we add the constraint $z_{v_0} \leq z_{v_m}$, then our (new, smaller) feasible set still contains an optimal solution. If we can find another Steinerpath in the opposite direction, from v_m to v_0, then we can equate the two variables, strictly reducing the size of the problem.

Steinerpaths were originally introduced in the context of a slightly different variant of the Steiner tree problem, in which a set of nodes $K \subseteq V_G$ is *required* to be included in the solution. Steinerpaths were required to emanate from K, but we see no need for such a requirement. One of our contributions is recognizing that Steinerpaths between any two nodes can be used to reduce the size of a NEWCS problem.

Finding Steinerpaths

Steinerpaths starting from a fixed set of nodes were originally found using a variant of Dijkstra's algorithm (Duin and Volgenant, 1987). Since we wish to discover Steinerpaths regardless of their source, we instead propose a cubic-time all-pairs Steinerpaths algorithm (Algorithm 2) based on the Floyd-Warshall all-pairs shortest path algorithm (Roy, 1959; Floyd, 1962; Warshall, 1962):

Algorithm 2 runs in $O(n^3)$ time, where $n = |V_G|$. Floyd-Warshall runs in cubic time, and the triply-nested for loop (Lines 4–8) also runs in cubic time. We restrict arc weights to only their nonpositive part (Line 2) because Floyd-Warshall assumes nonnegative distances as input, in order to return simple paths.[7] Because $w_{\leq 0}$ uses only the nonpositive part of arc weights, $-d_{s,t}$ is an underestimate of $w(p_{s,t})$, and in fact $-d_{s,t}$ is an underestimate of the weight of every suffix of $p_{s,t}$. Thus, Line 7 ensures that every suffix of the returned path has positive weight, and that the path is

[7]Even by allowing arcs of weight 0, the implementation requires some extra care to break ties in favor of simple paths.

Algorithm 2: All-Pairs Steinerpaths

```
Data: A graph $G$ with node and edge weights given by $w$.
Result: A set of Steinerpaths in $G$.
1 Let $G^*$ be the edge-weighted directed graph defined from $G$ as above.
2 Let $w_{\leq 0}$ be a new arc weight function, given by $w_{\leq 0}(a) = -\min(w(a), 0)$.
3 Run Floyd-Warshall on $G^*$ with arc distances given by $w_{\leq 0}$ to get, for all pairs $s, t$, the
    shortest path from $s$ to $t$. Call this shortest path $p_{s,t}$, and its distance $d_{s,t}$.
4 for $s \in V_G$ do
5     for $t \in V_G$ do
6         for $u \in V_G$ do
7             if $w(t \rightarrow u) \geq d_{s,t}$ and $u \notin p_{s,t}$ then
8                 yield $\langle p_{s,t}, t \rightarrow u \rangle$ ;
```

simple. Algorithm 2 is **precise**: every path returned is a Steinerpath. But the algorithm does not return every Steinerpath. In fact, exhaustively finding every Steinerpath is an intractable problem, which we prove by reduction from a known NP-complete problem:

Theorem 3. *Determining in general whether or not there exists a Steinerpath from s to t for $s, t \in V_G$ is NP-hard.*

Proof. We show that the problem of determining whether a graph K admits a Hamiltonion circuit with cost $\leq k$ can be reduced to finding a Steinerpath in a graph of size $|V_K| + 3$.

Given an edge-weighted graph K, arbitrarily choose a start vertex $r \in V_K$. Remove r and add two new vertices r_s and r_t. Each should get a copy of the same set of adjacent arcs that r had. Let M be some number greater than the sum of all edge weights in K. Give each node weight M. Also introduce two new nodes s and t with weight 0. Introduce an arc $s \rightarrow r_s$ with weight $-M(n+1)$. Introduce an arc $r_t \rightarrow t$ with weight k. There exists a Hamiltonion circuit in K with cost $\leq k$ if and only if there exists a Steinerpath from s to t. □

Not only is intractable to find every Steinerpath, the following lemma shows that even if we are given all Steinerpaths, they do not alone suffice to reduce the problem space to a single solution.

Lemma 5. *Optimal solutions are not necessarily covered by Steinerpaths.*

Proof. Let G be the graph illustrated in Figure 4.2. G is its own solution ($H = G$), but there are no Steinerpaths that include the edge $b \leftrightarrow e$. $\square$

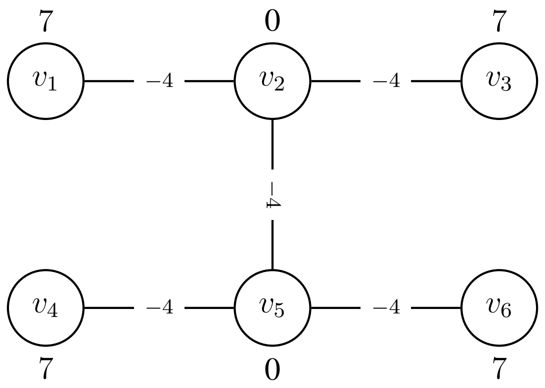


Figure 4.2: A graph G whose NEWCS solution is not covered by Steinerpaths. $\text{NEWCS}(G, w) = G$, and G's Steinerpaths are $\langle v_2 \to v_1 \rangle$, $\langle v_2 \to v_3 \rangle$, $\langle v_5 \to v_4 \rangle$, and $\langle v_5 \to v_6 \rangle$. The edge $v_2 \leftrightarrow v_5$ is not covered.

4.6 Other Applications in Natural Language Processing

So far, we have noted the potential applications of NEWCS to AMR and MRS parsing, as well as graph-based abstractive summarization. Several other semantic representations encode the meaning of a natural language sentence as a labeled directed connected graph, and so are potential applications. **Frame-semantic parses** were given in the form of a whole-sentence graph in one track of a 2007 shared task (Baker et al., 2007). Jointly predicting nodes and edges in any semantic dependency formalism is an instance of the NEWCS problem.

Joint *syntactic* dependency parsing and sentence compression can be posed as a PCST problem. The task has been solved exactly with an ILP (Thadani and McKeown, 2013), then later approximately using Lagrangian relaxation (Thadani, 2014). When parsing more informal forms of language, like that of social media or spoken dialog, some have noted that not every token participates in the syntactic analysis (*e.g.,* Charniak and Johnson, 2001; Foster et al., 2011). Similarly, some work suggests that **multi-word expressions** should be analyzed as a single unit in syntax (*e.g.,* Schneider et al., 2014). Jointly predicting syntactic dependency parses and excluded/merged tokens could be formulated as an instance of the PCST problem. Taking this approach would be most useful in languages with frequently non-projective syntax. Whereas semantic graphs tend to be non-projective, syntax for some languages (such as English) tends to be highly projective. For such languages, chart parsing techniques for projective parsing would be more appropriate, as they can be extended in polynomial time to jointly predict which nodes should be excluded or merged using the Bar-Hillel construction (Bar-Hillel et al., 1961).

4.7 Conclusion

We have introduced the node-and-edge-weighted connected subgraph (NEWCS) problem, an extension of the prize-collecting, and node-weighted, Steiner tree problems. We gave an ILP formulation as a baseline solver (§4.3), and a contributed a novel approximate algorithm (§4.4). We developed new reduction tests based on **Steinerpaths**, that can speed up solving for both NEWCS (and hence PCST) problems (§4.5). Finally, in Section 4.6, we noted other areas in NLP where predicting connected, but not necessarily spanning, graphs is required, suggesting that the potential impact of this work extends beyond semantic parsing.

Chapter 5

Encoding Networks

5.1 Introduction

Up to this point, this thesis has focused on how to *decode* linguistic graphs. Here, we turn our attention toward the problem of *encoding* graphs. We introduce a class of neural models for encoding graphs, called **Kleene Graph Encoder Networks (KGEN**). In contrast to **graph convolutional networks** (GCN; Kipf and Welling, 2016), KGENs take an edge-centric view, and encode *all* paths in a graph via dynamic programming. The encoding can be done efficiently and exactly for moderately sized graphs when we use composition functions that follow the star semiring laws (Lehmann, 1977). We hypothesize that by restricting the form of composition functions, KGENs are:

- able to capture longer-distance dependencies while maintaining efficient inference,
- able to respect graph *invariants*, such as invariance to node reordering, and
- interpretable.

This chapter is broken up into two main sections. The first (§5.2) shows that it is possible to define expressive composition functions that follow the star semiring laws. In particular, we introduce **Soft Patterns** (**SoPa**), a type of finite state automaton with neural weights, whose transitions can be constrained so that SoPa and its decisions are highly interpretable. Viterbi inference in weighted finite-state automata (WFSAs) follows the star semiring laws, and we use the model as a sequence encoder, experimenting on a suite of document classification tasks. We show that SoPa can be seen as a more powerful extension of a single-layer **convolutional neural network** (CNN; LeCun, 1998), but more efficient and interpretable (and less powerful) than recurrent neural networks like LSTMs (Hochreiter and Schmidhuber, 1997).

In the second section (§5.3), we show how such a model can be used to learn contextualized tensor representations of *graphs* and their parts. These representations can be used for a downstream task, and can be learned end-to-end in conjunction with that downstream task. We test the model's effectiveness on a semantic dependency parsing task. We compare to a strong baseline based on a graph convolutional network–based semantic role labeling system (Marcheggiani and Titov, 2017).

5.2 Sequence Encoding: SoPa

[*] Recurrent neural networks (RNNs; Elman, 1990) and convolutional neural networks (CNNs; LeCun, 1998) are two of the most useful text representation methods in NLP (Goldberg, 2016). These methods are generally considered to be quite different: the former encodes an arbitrarily long sequence of text, and is highly expressive (Siegelmann and Sontag, 1995). The latter is more local, encoding fixed length windows, and accordingly less expressive. In this paper, we seek to bridge the expressivity gap between RNNs and CNNs, presenting **SoPa** (for **So**ft **Pa**tterns), a model that lies in between them.

SoPa is a neural version of a weighted finite-state automaton (WFSA), with a restricted set of transitions. Linguistically, SoPa is appealing as it is able to capture a soft notion of surface patterns (e.g., *"what a great **X** !"*, Hearst, 1992), where some words may be dropped, inserted, or replaced with similar words (see Figure 5.1). From a modeling perspective, SoPa is interesting because

[*]*The work described in Section 5.2 was done with Roy Schwartz and Noah A. Smith, and published as Schwartz et al. (2018).*

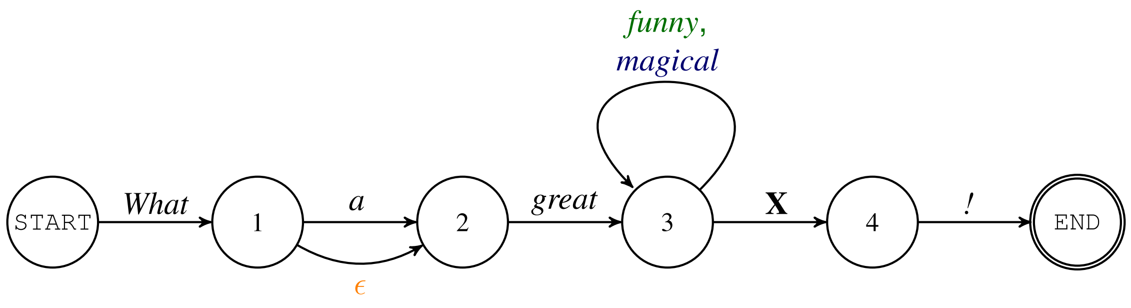


Figure 5.1: A representation of a surface pattern as a six-state automaton. **Self-loops** allow for repeatedly inserting words (e.g., "*funny*"). **ϵ-transitions** allow for dropping words (e.g., "*a*").

WFSAs are well-studied, and they come with efficient and flexible inference algorithms (*e.g.,* Mohri, 1997; Eisner, 2002).

SoPa defines a set of soft patterns of different lengths, with each pattern represented as a WFSA (Section 5.2.2). While the number and length of the patterns are hyperparameters, the patterns themselves are learned end-to-end. SoPa then represents a document with a vector that is the aggregate of the scores computed by matching each of the patterns with each span in the document.

We then show that SoPa is in fact a generalization of a one-layer CNN (Section 5.2.3). Accordingly, SoPa is more expressive than a one-layer CNN, and can encode flexible-length text spans. We show that such CNNs can be thought of as a collection of linear-chain WFSAs, each of which can only match fixed-length spans.

To test the utility of SoPa, we experiment with three text classification tasks (Section 5.2.4). We compare against four baselines, including both a bidirectional LSTM and a CNN. Our model performs on par or better than all baselines on all tasks (Section 5.2.5). Moreover, when training with smaller datasets, SoPa is particularly useful, outperforming all models by substantial margins. Finally, building on the connections discovered in this paper, we offer a new, simple method for the interpretation of SoPa (Section 5.2.6). This method applies equally well to CNNs.[1]

5.2.1 Background

Surface patterns. Patterns (Hearst, 1992) are particularly useful tool in NLP (*e.g.,* Lin et al., 2003; Etzioni et al., 2005; Schwartz et al., 2015). The most basic definition of a pattern is a sequence of words and wildcards (e.g., "**X** is a **Y**"), which can either be manually defined or extracted from corpus using cooccurrence statistics. Patterns are then used to be matched against a specific text span by replacing wildcards with concrete words.

Davidov et al. (2010) introduced a *flexible* notion of patterns, which supports partial matching of the pattern with a given text by skipping some of the words in the pattern, or introducing new words. In their framework, when a sequence of text partially matches a pattern, hard-coded partial scores are assigned to the pattern match. Here, we represent patterns as WFSAs with neural weights, and support these partial matches in a soft manner.

WFSAs. We review weighted finite-state automata with ϵ-transitions before we move on to our special case in Section 5.2.2. A WFSA-ϵ with d states over a vocabulary V is formally defined as a tuple $F = \langle \pi, \mathbf{T}, \eta \rangle$, where $\pi \in \mathbb{R}^d$ is an initial weight vector, $\mathbf{T} : (V \cup \{\epsilon\}) \to \mathbb{R}^{d \times d}$ is a transition weight function, and $\eta \in \mathbb{R}^d$ is a final weight vector. Given a sequence of words in the vocabulary $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$, the Forward algorithm (Baum and Petrie, 1966) scores $\boldsymbol{x}$ with respect to F. Without ϵ-transitions, Forward can be written as a series of matrix multiplications:

$$p'_{\text{span}}(\boldsymbol{x}) = \pi^\top \left(\prod_{i=1}^{n} \mathbf{T}(x_i) \right) \eta \tag{5.1}$$

ϵ-transitions are followed without consuming a word, so Equation 5.1 must be updated to reflect the possibility of following any number (zero or more) of ϵ-transitions in between consuming each

[1] We release our code at `http://anonymous`

word:

$$p_{\text{span}}(\boldsymbol{x}) = \pi^\top \mathbf{T}(\epsilon)^* \left(\prod_{i=1}^{n} \mathbf{T}(x_i) \mathbf{T}(\epsilon)^* \right) \eta \tag{5.2}$$

where * is matrix asteration: $A^* := \sum_{j=0}^{\infty} A^j$. In our experiments we use a first-order approximation, $A^* \approx I + A$, which corresponds to allowing zero or one ϵ-transition at a time. When the FSA F is probabilistic, the result of the Forward algorithm can be interpreted as the marginal probability of all paths through F while consuming $\boldsymbol{x}$ (hence the symbol "p").

The Forward algorithm can be generalized to any semiring (Eisner, 2002), a fact that we make use of in our experiments and analysis.[2] The vanilla version of Forward uses the probability semiring: $\oplus = +$, $\otimes = \times$. A special case of Forward is the Viterbi algorithm (Viterbi, 1967), which sets $\oplus = \max$. Viterbi finds the *highest scoring* path through F while consuming $\boldsymbol{x}$. Both Forward and Viterbi have runtime $O(d^3 + d^2 n)$, requiring just a single linear pass through the phrase. Using first-order approximate asteration, this runtime drops to $O(d^2 n)$.[3]

Finally, we note that Forward scores are for *exact matches*—the entire phrase must be consumed. We show in Section 5.2.2 how phrase-level scores can be summarized into a document-level score.

5.2.2 SoPa: A Weighted Finite-state Automaton RNN

We introduce SoPa, a WFSA-based RNN, which is designed to represent text as collection of surface pattern occurrences. We start by showing how a single pattern can be represented as a WFSA-ϵ (Section 5.2.2). Then we describe how to score a complete document using a pattern (Section 5.2.2). Finally, we describe how multiple patterns can be used to encode a document (Section 5.2.2).

[2]The semiring parsing view (Goodman, 1999) has produced unexpected connections in the past (Eisner, 2016). We experiment with max-times and max-plus semirings, but note that our model could be easily updated to use any semiring.

[3]In our case, we also use a sparse transition matrix (Section 5.2.2), which further reduces our runtime to $O(dn)$.

Patterns as WFSAs

We describe how a pattern can be represented as a WFSA-ϵ. We first assume a single pattern. A pattern *is* a WFSA-ϵ, but we impose hard constraints on its shape, and its transition weights are given by differentiable functions that have the power to capture concrete words, wildcards, and everything in between. Our model is designed to behave similarly to flexible hard patterns (see Section 5.2.1), but to be learnable directly and "end-to-end" through backpropagation. Importantly, it will still be interpretable as simple, almost linear-chain, WFSA-ϵ.

Each pattern has a sequence of d states (in our experiments we use patterns of varying lengths between 2 and 7). Each state i has exactly three possible outgoing transitions: a **self-loop**, which allows the pattern to consume a word without moving states, a **main path** transition to state $i + 1$ which allows the pattern to consume one token and move forward one state, and an **ϵ-transition** to state $i + 1$, which allows the pattern to move forward one state without consuming a token. All other transitions are given score 0. When processing a sequence of text with a pattern p, we start with a special `START` state, and only move forward (or stay put), until we reach the special `END` state.[4] A pattern with d states will tend to match token spans of length $d - 1$ (but possibly shorter spans due to ϵ-transitions, or longer spans due to self-loops). See Figure 5.1 for an illustration.

Our transition function, $\mathbf{T}$, is a parameterized function that returns a $d \times d$ matrix. For a word x:

$$[\mathbf{T}(x)]_{i,j} = \begin{cases} \sigma(\mathbf{u}_i \cdot \mathbf{v}_x + a_i), & \text{if } j = i \text{ \textit{(self-loop)}} \\ \sigma(\mathbf{w}_i \cdot \mathbf{v}_x + b_i), & \text{if } j = i + 1 \\ 0, & \text{otherwise,} \end{cases} \tag{5.3}$$

where $\mathbf{u}_i$ and $\mathbf{w}_i$ are vectors of parameters, a_i and b_i are scalar parameters, $\mathbf{v}_x$ is a fixed pre-trained word vector for x, and σ is the sigmoid function. ϵ-transitions are also parameterized, but don't

[4] To ensure that we start in the `START` state and end in the `END` state, we fix $\pi = [1, 0, \ldots, 0]$ and $\eta = [0, \ldots, 0, 1]$.

consume a token and depend only on the current state:

$$[\mathbf{T}(\epsilon)]_{i,j} = \begin{cases} \sigma(c_i), & \text{if } j = i+1 \\ 0, & \text{otherwise,} \end{cases} \tag{5.4}$$

where c_i is a scalar parameter.[5] As we have only three non-zero diagonals in total, the matrix multiplications in Equation 5.2 can be implemented using vector operations, and the overall runtimes of Forward and Viterbi are reduced to $O(dn)$.[6]

Words vs. wildcards. Traditional *hard* patterns distinguish between words and wildcards. Our model does not explicitly capture the notion of either, but the transition weight function is simple enough that it can be interpreted in those terms.

Each transition is a logistic regression over the next word vector $\mathbf{v}_x$. For example, for a main path out of state i, $\mathbf{T}$ has two parameters, $\mathbf{w}_i$ and b_i. If $\mathbf{w}_i$ has large magnitude and is close to the word vector for some word y (e.g., $\mathbf{w}_i \approx 100\mathbf{v}_y$), and b_i is a large negative bias (e.g., $b_i \approx -100$), then the transition is essentially matching the specific word y. Whereas if $\mathbf{w}_i$ has small magnitude ($\mathbf{w}_i \approx \mathbf{0}$) and b_i is a large positive bias (e.g., $b_i \approx 100$), then the transition is ignoring the current token and matching a wildcard.[7] The transition could also be something in between, for instance by focusing on specific dimensions of a word's meaning encoded in the vector, such as POS or semantic features like animacy or concreteness (Rubinstein et al., 2015; Tsvetkov et al., 2015).

Scoring Documents

So far we described how to calculate how well a pattern matches a token span exactly (consuming the whole span). To score a complete document, we prefer a score that aggregates over all matches

[5] Adding ϵ-transitions to WFSAs does not increase their expressive power, and in fact slightly complicates the Forward equations. We use them as they require fewer parameters, and make the modeling connection between (hard) flexible patterns and our (soft) patterns more direct and intuitive.

[6] Our implementation is optimized to run on GPUs, so the observed runtime is even *sublinear* in d.

[7] A large bias increases the eagerness to match *any* word.

on subspans of the document (similar to "search" instead of "match", in regular expression parlance). We still assume a single pattern.

Either the Forward algorithm can be used to calculate $p_{\text{doc}}(\boldsymbol{x}) = \sum_{1 \leq i \leq j \leq n} p_{\text{span}}(\boldsymbol{x}_{i:j})$,[8] or Viterbi to calculate $s_{\text{doc}}(\boldsymbol{x}) = \max_{1 \leq i \leq j \leq n} s_{\text{span}}(\boldsymbol{x}_{i:j})$.[9] In short documents, we expect patterns to typically occur at most once, so we choose the Viterbi algorithm in our experiments.

Implementation details. In our experiments we use the Viterbi inference in the probability semiring (i.e., $\oplus = \max, \otimes = \times$). We give the specific recurrences we use to score documents in a single pass with this model. We define:

$$[\text{maxmul}(\mathbf{A}, \mathbf{B})]_{i,j} = \max_k A_{i,k} B_{k,j}. \tag{5.5}$$

We also define the following for taking zero or one ϵ-transitions:

$$\text{eps}(\mathbf{v}) = \text{maxmul}(\mathbf{v}, \max(\mathbf{I}, \mathbf{T}(\epsilon))) \tag{5.6}$$

We keep a state vector $\mathbf{h}_t$ at each token:

$$\mathbf{h}_0 = \text{eps}(\pi^\top) \tag{5.7a}$$

$$\mathbf{h}_{t+1} = \max(\text{eps}(\text{maxmul}(\mathbf{h}_t, \mathbf{T}(x_{t+1}))), \mathbf{h}_0) \tag{5.7b}$$

$$s_t = \text{maxmul}(\mathbf{h}_t, \eta) \tag{5.7c}$$

$$s_{\text{doc}} = \max_{1 \leq t \leq n} s_t \tag{5.7d}$$

$[\mathbf{h}_t]_i$ represents the score of the best path through the pattern that ends in state i after consuming t tokens. By including $\mathbf{h}_0$ in Equation 5.7b, we are accounting for spans that start at time t. s_t is the

[8]With a probabilistic WFSA, this is the expected count of the pattern in the document.

[9]This is the score of the highest-scoring match.

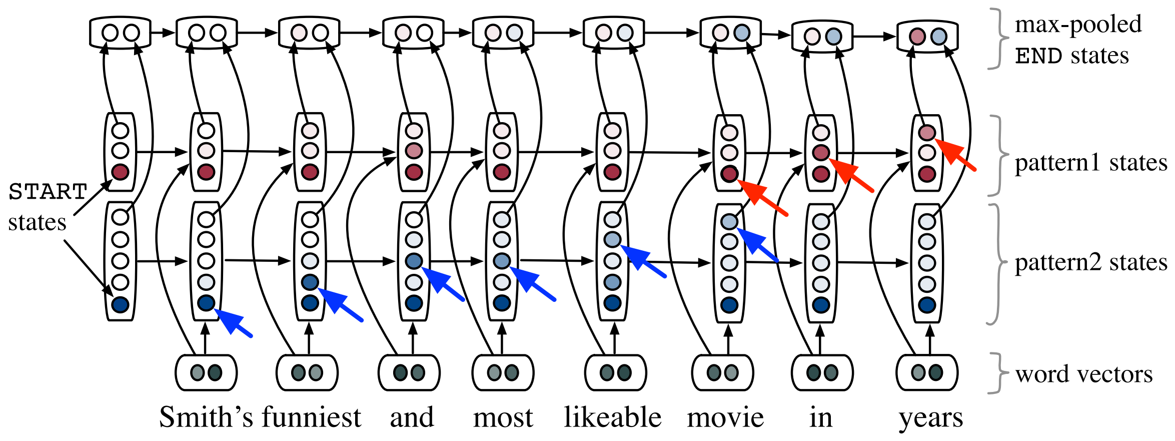


Figure 5.2: State activations of two patterns as they score a document. pattern1 (length three) matches on *'in years'*. pattern2 (length five) matches on *'funniest and most likeable movie'*, using a self-loop to consume the token *'most'*. Active states in the best match are marked with arrow cursors.

maximum of the exact match scores for all spans ending at token t. And s_{doc} is the maximum score of any subspan in the document.

Aggregating Multiple Patterns

We describe how k patterns are aggregated to score a document. These k patterns give k different s_{doc} scores for the document, which are stacked into a vector $\mathbf{z} \in \mathbb{R}^k$ and constitute the final document representation of SoPa. This vector representation can be viewed as a feature vector. In this paper, we feed it into a multilayer perceptron (MLP), culminating in a softmax to give a probability distribution over document labels. We minimize cross-entropy, allowing the SoPa and MLP parameters to be learned end-to-end.

SoPa uses a total of $(2e + 3)dk$ parameters, where e is the word embedding dimension, d is the number of states and k is the number of patterns. For comparison, an LSTM with a hidden dimension of h has $4((e + 1)h + h^2)$. In Section 5.2.5 we show that SoPa consistently uses fewer parameters than a BiLSTM baseline to achieve its best result.

SoPa as an RNN. SoPa can in fact also be considered as a simple RNN. As shown above, a pattern with d states has a hidden state of size d. Stacking the k hidden states of k patterns into one vector of size $k \times d$ can be thought of as the hidden state of our model. This hidden state is, like in any other RNN, dependent of the input and the previous state. Using self-loops, the hidden state at time point i can in theory depend on the entire document (see Figure 5.2 for illustration). Importantly, since all transitions pass through a sigmoid and so have scores strictly less than 1, our model is discouraged from following self-loops, only following them if it results in a better fit with the remainder of the pattern.[10]

Although even single-layer RNNs are Turing complete (Siegelmann and Sontag, 1995), SoPa's expressive power is limited and determined by the semiring $\mathcal{K}$ used during the Forward algorithm. When a WFSA is thought of as a function from finite sequences of tokens to $\mathcal{K}$, it is restricted to the class of functions known as **rational series** (Schützenberger, 1961; Droste and Gastin, 1999; Sakarovitch, 2009).[11] It is unclear how limiting this theoretical restriction is in practice, especially when SoPa is used as a component in a larger network. Next, we show that SoPa is a generalization of a one-layer CNN, thus more expressive.

5.2.3 CNN as SoPa

A convolutional neural network (**CNN**; LeCun, 1998) moves a fixed-size sliding window over the document, producing a vector representation for each window. These representations are then often summed, averaged, or max-pooled to produce a document-level representation (*e.g.,* Kim, 2014; Yin and Schütze, 2015). In this section, we show that SoPa is in fact a generalization of one-layer CNNs.

To recover a CNN from a soft pattern with $d + 1$ states, we remove self-loops and ϵ-transitions, retaining only the main path transitions. We also remove the sigmoid in the main path transition

[10]Using other semirings can result in different dynamics, potentially encouraging rather than discouraging self-loops.

[11]Rational series generalize recognizers of *regular languages*, which are the special case where $\mathcal{K}$ is the Boolean semiring.

score (Equation 5.3), and use the max-sum semiring when running Forward. With only main path transitions, the network will not match any span that is not exactly d tokens long. Using max-sum, spans of length d will be assigned the score:

$$s_{\text{span}}(\boldsymbol{x}_{i:i+d}) = \sum_{j=0}^{d-1} \mathbf{w}_j \cdot \mathbf{v}_{x_{i+j}} + b_j, \tag{5.8a}$$

$$= \mathbf{w}_{0:d} \cdot \mathbf{v}_{x_{i:i+d}} + \sum_{j=0}^{d-1} b_j, \tag{5.8b}$$

where $\mathbf{w}_{0:d} = [\mathbf{w}_0^\top; \ldots ; \mathbf{w}_{d-1}^\top]^\top$, $\mathbf{v}_{x_{i:i+d}} = [\mathbf{v}_{x_i}^\top; \ldots ; \mathbf{v}_{x_{i+d-1}}^\top]^\top$. We can now recognize this as an affine transformation of the concatenated word vectors $\mathbf{v}_{x_{i:i+d}}$, which is just the same as a linear convolutional filter with window size d.[12] A single pattern's score for a document is:

$$s_{\text{doc}}(\boldsymbol{x}) = \max_{1 \leq i \leq n-d+1} s_{\text{span}}(\boldsymbol{x}_{i:i+d}). \tag{5.9}$$

Using k patterns with $d+1$ states corresponds to a CNN with window size d and output dimension k. The $\max$ in Equation 5.9 is calculated for each pattern independently, corresponding to element-wise max-pooling of the CNN's output layer. Based on the equivalence between this impoverished version of SoPa and CNNs, we conclude that one-layer CNNs are learning an even more restricted class of WFSAs (linear-chain WFSAs) that capture only fixed-length patterns.

One notable difference between SoPa and CNNs is that CNNs can use any subdifferentiable nonlinear filter (like an MLP over $\mathbf{v}_{x_{i:i+d}}$, for example). In contrast, in order to efficiently pool over flexible-length spans, SoPa is restricted to operations that follow the semiring laws.[13]

As a model that is more flexible than a one-layer CNN, but less expressive than an RNN, SoPa lies somewhere on the continuum between these two approaches. Continuing to study the bridge

[12]This variant of SoPa has d bias parameters, which correspond to only a single bias parameter in a CNN. The redundant biases may affect optimization but are an otherwise unimportant difference.

[13]The max-sum semiring corresponds to a linear filter. Other semirings could potentially model more interesting interactions, but we leave this to future work.

between CNNs and RNNs is an exciting direction for future research.

5.2.4 Experiments

To evaluate SoPa, we apply it to text classification tasks. Below we describe our datasets and baselines. More details can be found in Appendix A.2.

Datasets. We experiment with three binary classification datasets.

- **SST**. The Stanford Sentiment Treebank (Socher et al., 2013)[14] contains roughly 10K movie reviews from Rotten Tomatoes,[15] labeled on a scale of 1–5. We consider the binary task, which considers 1 and 2 as negative, and 4 and 5 as positive (ignoring 3s). It is worth noting that this dataset also contains syntactic phrase level annotations, providing a sentiment label to parts of sentences. In order to experiment in a realistic setup, we only consider the complete sentences, and ignore syntactic annotations at train or test time. The number of training/development/test sentences in the dataset is 6,920/872/1,821.
- **Amazon**. The Amazon Review Corpus (McAuley and Leskovec, 2013)[16] contains electronics product reviews, a subset of a larger review dataset. Each document in the dataset contains a review and a summary. Following Yogatama et al. (2015), we only use the reviews part, focusing on positive and negative reviews. The number of training/development/test samples is 20K/5K/25K.
- **ROC**. The ROC story cloze task (Mostafazadeh et al., 2016) is a story understanding task.[17] The task is composed of four-sentence story prefixes, followed by two competing endings: one that makes the joint five-sentence story coherent, and another that makes it incoherent. Following Schwartz et al. (2017), we treat it as a style detection task: we treat all "right"

[14] https://nlp.stanford.edu/sentiment/index.html
[15] http://www.rottentomatoes.com
[16] http://riejohnson.com/cnn_data.html
[17] http://cs.rochester.edu/nlp/rocstories/

endings as positive samples and all “wrong” ones as negative, and we ignore the story prefix. We split the development set into train and development (of sizes 3,366 and 374 sentences, respectively), and take the test set as-is (3,742 sentences).

Reduced training data. In order to test our model’s ability to learn from small datasets, we also randomly sample 100, 500, 1,000 and 2,500 SST training instances and 100, 500, 1,000, 2,500, 5,000, and 10,000 Amazon training instances. Development and test sets remain the same.

Baselines. We compare to four baselines: a BiLSTM, a one-layer CNN, DAN (a simple alternative to RNNs) and a feature-based classifier trained with hard-pattern features.

- **BiLSTM.** Bidirectional LSTMs have been successfully used in the past for text classification tasks (Zhou et al., 2016). We learn a one-layer BiLSTM representation of the document, and feed the average of all hidden states to an MLP.
- **CNN**. CNNs are particularly useful for text classification (Kim, 2014). We train a one-layer CNN with max-pooling, and feed the resulting representation to an MLP.
- **DAN**. We learn a deep averaging network with word dropout (Iyyer et al., 2015a), a simple but strong text-classification baseline.
- **Hard**. We train a logistic regression classifier with hard-pattern features. Following Tsur et al. (2010), we replace low frequency words with a special wildcard symbol. We learn sequences of 1–6 concrete words, where any number of wildcards can come between two adjacent words. We consider words occurring with frequency of at least 0.01% of our training set as concrete words, and words occurring in frequency 1% or less as wildcards.[18]

Number of patterns. SoPa requires specifying the number of patterns to be learned, and their lengths. Preliminary experiments showed that the model doesn’t benefit from more than a few

[18]Some words may serve as both words and wildcards. See Davidov and Rappoport (2008) for discussion.

Model	**ROC**	**SST**	**Amazon**
Hard	62.2 (4K)	75.5 (6K)	88.5 (67K)
DAN	64.3 (91K)	83.1 (91K)	85.4 (91K)
BiLSTM	65.2 (844K)	84.8 (1.5M)	**90.8** (844K)
CNN	64.3 (155K)	82.2 (62K)	90.2 (305K)
SoPa	**66.5** (255K)	**85.6** (255K)	90.5 (256K)
$ms \backslash \{\sigma\}$	64.4	84.8	90.0
$ms \backslash \{\sigma, sl\}$	63.2	84.6	89.8
$ms \backslash \{\sigma, \epsilon\}$	64.3	83.6	89.7
$ms \backslash \{\sigma, sl, \epsilon\}$	64.0	85.0	89.5

Table 5.1: Test classification accuracy (and the number of parameters used). The bottom part shows our ablation results: SoPa: our full model. ms: running with max-sum semiring (rather than max-times). sl: self-loops, ϵ: ϵ transitions, σ: sigmoid transition (see Equation 5.3). The final row is equivalent to a one-layer CNN.

dozen patterns. We experiment with several configurations of patterns of different lengths, generally considering 0, 10 or 20 patterns of each pattern length between 2–7. The total number of patterns learned ranges between 30–70.[19]

5.2.5 Results

Table 5.1 shows our main experimental results. In two of the cases (SST and ROC), SoPa outperforms all models. On Amazon, SoPa performs within 0.3 points of CNN and BiLSTM, and outperforms the other two baselines. The table also shows the number of parameters used by each model for each task. Given enough data, models with more parameters should be expected to perform better. However, SoPa performs better or roughly the same as a BiLSTM, which has 3–6 times as many parameters.

Figure 5.3 shows a comparison of all models on the SST and Amazon datasets with varying training set sizes. SoPa is substantially outperforming all baselines, in particular BiLSTM, on small

[19]The number of patterns and their length are hyperparameters tuned on the development data (see Appendix A.2).

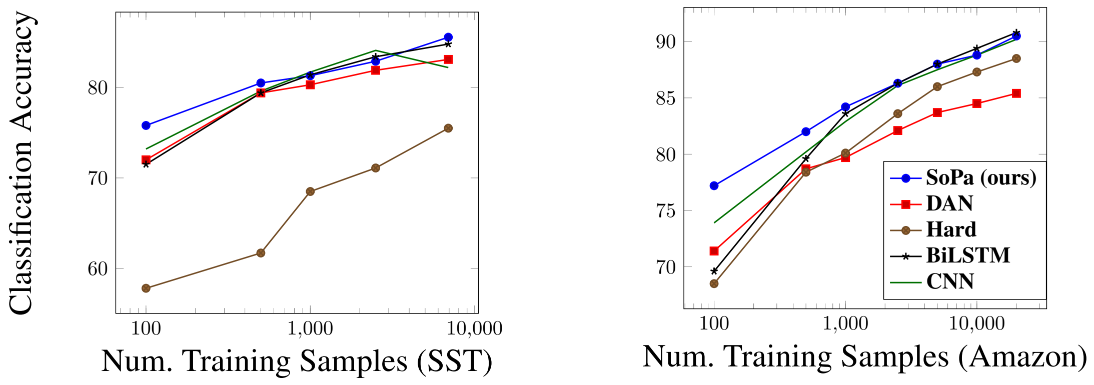


Figure 5.3: Test accuracy on SST and Amazon with varying number of training instances.

datasets (100 samples). This suggests that SoPa, being less expressive than BiLSTM, is better fit to learn from small datasets.

Ablation analysis. Table 5.1 also shows an ablation of the differences between SoPa and CNN: max-times semiring with sigmoid vs. max-sum semiring, self-loops, and ϵ-transitions. The last line is equivalent to a CNN with multiple window sizes. Interestingly, the most notable difference between SoPa and CNN is the semiring, while ϵ transitions and self-loops have little effect on performance.[20]

5.2.6 Interpretability

We turn to another key aspect of SoPa—its interpretability. We start by demonstrating how we interpret a single pattern, and then describe how to interpret the decisions made by downstream classifiers that rely on SoPa—in this case, a sentence classifier. Importantly, these visualization techniques are equally applicable to CNNs.

Interpreting a single pattern. In order to visualize a pattern, we compute the pattern matching scores with each phrase in our training dataset, and select the k phrases with the highest scores. Table 5.2 shows examples of six patterns learned using the best SoPa model on the SST dataset,

[20]Although SoPa does make use of them—see Section 5.2.6.

as represented by their five highest scoring phrases in the training set. A few interesting trends can be observed from these examples. First, it seems our patterns encode semantically coherent expressions. A large portion of them correspond to sentiment (the five top examples in the table), but others capture different semantics, e.g., time expressions.

Second, it seems our patterns are relatively soft, and allow lexical flexibility. While some patterns do seem to fix specific words, e.g., "of" in the first example or "minutes" in the last one, even in those cases some of the top matching spans replace these words with other, similar words ("with" and "half-hour", respectively). Encouraging SoPa to have more concrete words, e.g., by jointly learning the word vectors, might make SoPa useful in other contexts, particularly as a decoder. We defer this direction to future work.

Finally, SoPa makes limited but non-negligible use of self-loops and epsilon steps. Interestingly, the second example shows that one of the patterns had an ϵ-transition at the same place in every phrase. This demonstrates a different function of ϵ-transitions than originally designed—they allow a pattern to effectively shorten itself, by learning a high ϵ-transition parameter for a certain state.

Interpreting a document. SoPa provides an interpretable representation of a document—a vector of the maximal matching score of each pattern with any span in the document. To visualize the decisions of our model for a given document, we can observe the patterns and corresponding phrases that score highly within it.

To understand which of the k patterns contributes most to the classification decision, we apply a leave-one-out method. We run the forward method of the MLP layer in SoPa k times, each time zeroing-out the score of a different pattern p. The difference between the resulting score and the original model score is considered p's contribution. Table 5.3 shows example texts along with their most positive and negative contributing phrases.

5.2.7 Related Work

Weighted finite-state automata. WFSAs and hidden Markov models[21] were once popular in automatic speech recognition (*e.g.,* Hetherington, 2004; Moore et al., 2006; Hoffmeister et al., 2012) and remain popular in morphology (*e.g.,* Dreyer, 2011; Cotterell et al., 2015). Most closely related to this work, neural networks have been combined with weighted finite-state transducers to do morphological reinflection (Rastogi et al., 2016). These prior works learn a single FSA or FST, whereas our model learns a collection of simple but complementary FSAs, together encoding a sequence. We are the first to incorporate neural networks both *before* WFSAs (in their transition scoring functions), and *after* (in the function that turns their vector of scores into a final prediction), to produce an expressive model that remains interpretable.

Recurrent neural networks. The ability of RNNs to represent arbitrarily long sequences of embedded tokens has made them attractive to NLP researchers. The most notable variants, the long short-term memory (LSTM; Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014), have become ubiquitous in NLP algorithms (Goldberg, 2016). Recently, several works introduced simpler versions of RNNs, such as recurrent additive networks (Lee et al., 2017) and Quasi-RNNs (Bradbury et al., 2017). Like SoPa, these models can be seen as points along the bridge between RNNs and CNNs.

Other works have studied the expressive power of RNNs, in particular in the context of WFSAs or HMMs (Cleeremans et al., 1989; Giles et al., 1992; Visser et al., 2001; Chen et al., 2018). In this work we have shown that one-layer CNNs are also in fact a restricted form of a WFSA.

Convolutional neural networks. CNNs are prominent feature extractors in NLP, both for generating character-based embeddings (Kim et al., 2016a), and as sentence encoders for tasks like text classification (Yin and Schütze, 2015) and machine translation (Gehring et al., 2017). Similarly

[21] HMMs are a special case of WFSAs (Mohri et al., 2002).

to SoPa, several recently introduced variants of CNNs support varying window sizes by either allowing several fixed window sizes (Yin and Schütze, 2015) or by supporting non-consecutive n-gram matching (Lei et al., 2015; Nguyen and Grishman, 2016).

Neural networks and patterns. Some works used patterns as part of a neural network. Schwartz et al. (2016) used pattern contexts for estimating word embeddings, showing improved word similarity results compared to bag-of-word contexts. Shwartz et al. (2016) designed an LSTM representation for dependency patterns, using them to detect hypernymy relations. Here, we learn patterns as a neural version of WFSAs.

Interpretability. There have been several efforts to interpret neural models. The weights of the attention mechanism (Bahdanau et al., 2015) are often used to display the words that are most significant for making a prediction. LIME (Ribeiro et al., 2016) is another approach for visualizing neural models (not necessarily textual). Yogatama and Smith (2014) introduced structured sparsity, which encodes linguistic information into the regularization of a model, thus allowing to visualize the contribution of different bag-of-word features.

Other works jointly learned to encode text and extract the span which best explains the model's prediction (Yessenalina et al., 2010; Lei et al., 2016). Li et al. (2016) suggested a method that erases pieces of the text in order to analyze their effect on a neural model's decisions. Finally, several works presented methods to visualize deep CNNs (Zeiler and Fergus, 2014; Simonyan et al., 2014; Yosinski et al., 2015), focusing on visualizing the different layers of the network, mainly in the context of image and video understanding. We believe these two types of research approaches are complementary: inventing general purpose visualization tools for existing black-box models on the one hand, and on the other, designing models like SoPa that are interpretable by construction.

5.2.8 Conclusion

We introduced SoPa—a novel model that combines neural representation learning with WFSAs. We showed that SoPa is a generalization of a one-layer CNN. It naturally models flexible-length spans with insertion and deletion, and it can be easily customized by swapping in different semirings. SoPa performs on par or strictly better than four baselines on three text classification tasks, while requiring fewer parameters than the stronger baselines. On smaller training sets, SoPa outperforms all four baselines. As a simple version of an RNN, which is more expressive than one-layer CNNs, we hope that SoPa will encourage future research on the bridge between these two mechanisms. To facilitate such research, we release our implementation at `https://github.com/Noahs-ARK/soft_patterns`.

Pattern	**Highest Scoring Phrases**				
Pattern 1	thoughtful	,	reverent	portrait	of
	and	astonishingly	articulate	cast	of
	entertaining	,	thought-provoking	film	with
	gentle	,	mesmerizing	portrait	of
	poignant	and	uplifting	story	in
Pattern 2	's	ϵ	uninspired	story	.
	this	ϵ	bad	on	purpose
	this	ϵ	leaden	comedy	.
	a	ϵ	half-assed	film	.
	is	ϵ	clumsy $,_{SL}$	the	writing
Pattern 3	mesmerizing	portrait	of	a	
	engrossing	portrait	of	a	
	clear-eyed	portrait	of	an	
	fascinating	portrait	of	a	
	self-assured	portrait	of	small	
Pattern 4	honest	,	and	enjoyable	
	soulful	, scathing_{SL}	and	joyous	
	unpretentious	, charming_{SL}	,	quirky	
	forceful	,	and	beautifully	
	energetic	,	and	surprisingly	
Pattern 5	is	deadly	dull		
	a	numbingly	dull		
	is	remarkably	dull		
	is	a	phlegmatic		
	an	utterly	incompetent		
Pattern 6	five	minutes			
	four	minutes			
	final	minutes			
	first	half-hour			
	fifteen	minutes			

Table 5.2: Six patterns of different lengths learned by SoPa on SST. Each group represents a single pattern p, and shows the five phrases in the training data that have the highest score for p. Columns represent pattern states. Words marked with $_{SL}$ are self-loops. ϵ symbols indicate ϵ-transitions. All other words are from main path transitions.

Analyzed Documents
it 's dumb , **but more importantly** , *it 's just not scary*
though moonlight mile is replete with **acclaimed actors and actresses** and tackles a subject that 's **potentially moving** , the movie is *too predictable* and *too self-conscious to reach a* level of **high drama**
While **its careful pace and** seemingly *opaque story* may not satisfy every moviegoer 's appetite, the film 's final scene is **soaringly , transparently moving**
unlike the speedy wham-bam effect *of most hollywood offerings* , character development – and more **importantly, character empathy – is at the heart of** italian for beginners .
the band 's courage in the face of official repression **is inspiring** , **especially for** aging *hippies* (this one included) .

Table 5.3: Documents from the SST training data. Phrases with the largest contribution toward a positive sentiment classification are in **bold green**, and the most negative phrases are in *italic orange*.

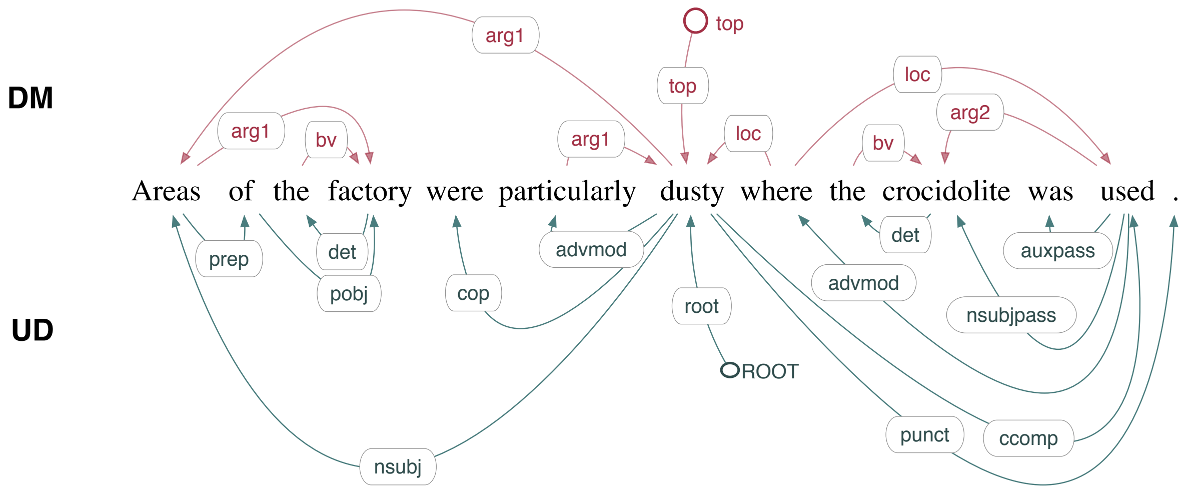


Figure 5.4: An example sentence annotated with semantic dependencies from **DM** (Oepen et al., 2014, *top/red*) and syntax from **Universal Dependencies** (Silveira et al., 2014, *bottom/green*).

5.3 Graph Encoding: Kleene Graph Encoder Networks

* In this section, we introduce a new class of graph neural networks, called **Kleene Graph Encoder Networks (KGEN)**. KGEN is a model architecture that generalizes SoPa ˜(§5.2), and extends it in order to encode *graphs* instead of sequences. In the same way that an RNN recursively propagates information forward (or backward) in a sentence through adjacent tokens, graph neural networks propagate information between adjacent nodes in a graph. Graphs are extremely general, and a good general purpose graph encoder has many potential uses. Graph convolutional networks, for example, were originally introduced in order to do semi-supervised learning on citation networks and on a knowledge graphs (Defferrard et al., 2016; Kipf and Welling, 2016), but have since been used for a variety of tasks, including machine translation (Bastings et al., 2017), named entity recognition (Cetoli et al., 2017), semantic role labeling (Marcheggiani and Titov, 2017), and multi-document summarization (Yasunaga et al., 2017), among others.

While most existing graph neural networks learn contextualized representations of *nodes*, our model learns contextualized representations of *paths*. We argue that this is a finer-grained level of representation (from which node representations can be recovered), and so has more potential uses.

The work described in Section 5.3 was done with Hao Peng, Jan Buys, and Noah A. Smith.

It is an especially natural level of representation when used for the task of predicting arcs over the same set of nodes as the input. We describe the most general form of the model architecture (§5.3.3) in terms of star semirings, an algebraic view of the model and accompanying algorithm which draws inspiration from prior work in NLP on semiring parsing (Goodman, 1999). We then describe the specific model we experiment with, which uses the maxmul operation defined in the previous section (Equation 5.5).

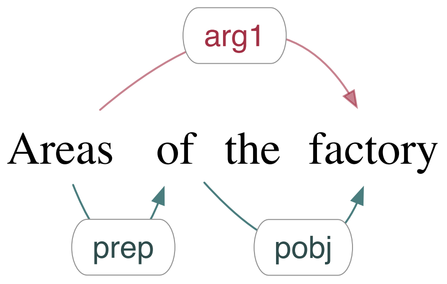
We test our model by encoding a syntactic dependency graph, and using the resulting representations to predict semantic dependencies (Oepen et al., 2014, 2015). Our experiments compare our model, KGEN, to a GCN baseline, and a syntax-free baseline on semantic dependency parsing (§5.3.4).

We make the following contributions:

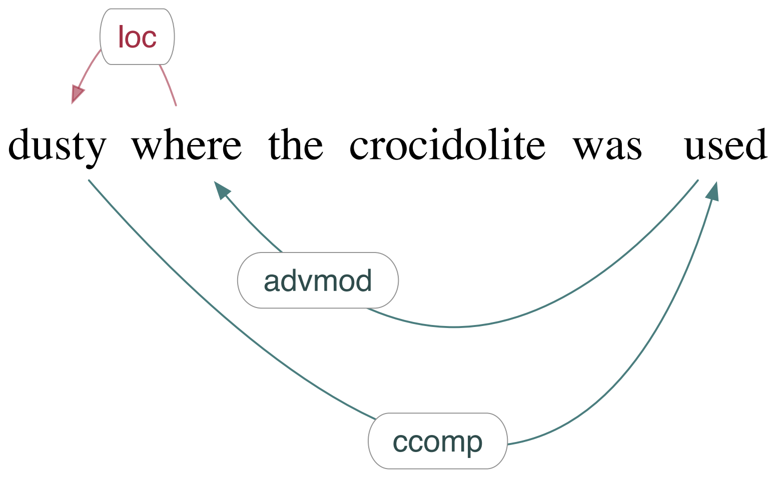
- We introduce **KGEN**, a new class of graph neural networks that captures features of *all paths* in an input graph, efficiently, using a dynamic program.
- We are the first to formulate a machine learning algorithm in terms of **star semirings**, expanding on the long line of semiring parsing work in NLP (Goodman, 1999; Eisner et al., 2004). We show how the *star* in star semirings can be used to calculate otherwise infinitely loopy computations in cubic time.
- We show experimentally that KGEN is effective at capturing both syntactic and linear order context, improving a semantic dependency parser more than a graph convolutional network baseline.

5.3.1 Related Work

Encoding nodes Graph convolutional networks have been used on citation networks and on a knowledge graphs (Defferrard et al., 2016; Kipf and Welling, 2016), machine translation (Bastings et al., 2017), named entity recognition (Cetoli et al., 2017), semantic role labeling (Marcheggiani



(a) An ARG1 DM arc corresponding to a length 2 syntactic path.



(b) A LOC DM arc corresponding to a length 2 syntactic path.

Figure 5.5: We highlight two DM arcs that do not have a directly corresponding syntactic arc, but do have a corresponding *path* of syntactic arcs.

and Titov, 2017), and multi-document summarization (Yasunaga et al., 2017), among others. GCNs provide contextualized *node* representations, which can be influenced by other nodes within a fixed distance, k. In exchange for limiting the distance, GCNs can use non-lawful composition functions. They can be thought of as encoding, in a node v's vector, all paths emanating from v (and ending anywhere) that have length exactly k. KGEN on the other hand, encodes, in a representation for each pair of nodes, $\mathbf{f}(u, v)$, all paths from u to v of any length. KGEN's is a finer-grained representation, because outgoing representations from u can be always pooled, $\max_v \mathbf{f}(u, v)$, to get a node representation analogous to GCN's. It is unclear that a GCN node representation for u can be "unpooled" to recover a representation for each opposite node v.

Encoding paths Paths of syntactic dependencies have long been recognized as useful features for predicting semantic relations. Conveniently, syntactic dependencies form an arborescence, so there is a unique path from any given source token to any other given destination token, making feature extraction easy and efficient. In Chapter 2, we experimented with several variants of hand-engineered path features (see Table 2.2), and found them to be some of the most predictive features.

Roth and Lapata (2016) were the first to automatically learn syntactic dependency path features with neural nets. Their semantic role labeling model ran an LSTM on the path between every candidate pair of nodes, and was learned end-to-end. This strategy works well for arborescences, which are relatively sparse, and where most of the predictive information lives on the direct shortest path between two nodes. But is it possible to capture information that lives *outside* the shortest path? What if there is no shortest path, or there are multiple? This situation arises as more NLP preprocessing is conditioned on. . . for instance, one could add predicted discourse trees, or named entity recognition, and then automatically learn features from the union of all the preprocessed structures. KGEN encodes and aggregates *all* paths between every pair of nodes (of any length, even paths with loops), so it is able to handle multigraphs, and capture "off the beaten path" information. Because the recurrence of our path-RNN and the pooling function we use to aggregate multiple paths together follow algebraic laws (§5.3.2), we are able to calculate and aggregate encodings for all paths efficiently.

Along this line, Toutanova et al. (2016) introduced a model for knowledge base completion that uses a *bilinear* composition function on paths. Their model includes a dynamic program to score all paths up to a fixed length. Our method generalizes and improves on (Toutanova et al., 2016) in a few key ways:

- Instead of calculating and aggregating scalar scores directly for paths, we calculate tensor representations. Arbitrary (i.e., non-linear) downstream modules can then be used on the output of KGEN, allowing for more powerful function learning without losing efficient inference.
- We relax the *bilinear* requirement, requiring only that the recurrence and pooling functions together follow the *semiring* laws (§5.3.2). Bilinear functions are a special case under the sum-product semiring, but we experiment with other useful semirings.
- Specifically, by working with a max-pooling semiring instead of a sum-pooling one, we have

a natural extension into a *star semiring* without fear of overflows. We can then encode all paths of *any* length between a given pair of nodes, even loopy paths. We hypothesize that this is important for encoding graph structures that deviate from the shortest path between two nodes.

Invariance to node reorderings Our approach and GCNs have the advantage that they can be agnostic to node order when desired. Nodes are unordered by definition in general graphs.[1] Deep Sets (Zaheer et al., 2017) studied the similar goal of being invariant to element ordering when encoding sets. They argue that encoding elements into $\mathbb{R}^d$, sum-pooling, then running an arbitrary non-linear function such as an MLP can be a universal function approximator on subsets, when d is at least the size of the set to be encoded. Follow up work showed d *must* be as large as the set (Wagstaff et al., 2019). We note that because semiring $\oplus$ is commutative and associative by law, this same goal is accomplished. Attention is also order invariant, a property which graph attention networks (Veličković et al., 2018) make use of. These are all special cases of invariance under group actions (Bloem-Reddy and Teh, 2019), the group action being permutation.

Another approach to contextualizing a node is to linearize its neighbors, and then run an (order-sensitive) RNN over them (Peng et al., 2017b; Zayats and Ostendorf, 2017).[2] An ordering of the nodes must be chosen though, even if there is no natural ordering.[3] Instead of choosing just one ordering, multiple permutations could be sampled, as in Janossy pooling (Murphy et al., 2018), but this comes at a high computational cost and only achieves approximate order invariance. We believe it is worth exploring approaches that are order invariant by construction, and finding the expressive limits of such approaches.

[1]One can always opt in to order-sensitivity by adding additional synthetic arcs, as in the "+" arcs in Figure 1.1 and the *"next"* arcs in Figure 5.6.

[2]An extreme version of this is to linearize the entire graph and encode it with an RNN (Niepert et al., 2016; Konstas et al., 2017; Zellers et al., 2018).

[3]In Zellers et al. (2018) we found that our model for scene graph parsing was relatively robust to ordering schemes, but we did observe differences.

5.3.2 Background: Star Semirings and Kleene's Algorithm

A star semiring is an algebraic structure which is closely connected to regular expressions and weighted finite state automata. Formally, it is a six-tuple, $(\mathcal{S}, ⓪, ①, \oplus, \otimes, ^{⊛})$, where $\mathcal{S}$ is a set, $⓪, ① \in \mathcal{S}$ are distinguished elements, $\oplus,\ \otimes : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ are binary operations, and $^{⊛} : \mathcal{S} \to \mathcal{S}$ is a unary operation, subject to the following laws. For all $a, b, c \in \mathcal{S}$:

$$⓪ \oplus a = a \oplus ⓪ = a \qquad \text{(⓪ is the identity for } \oplus\text{)} \tag{5.10a}$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \qquad \text{(}\oplus\text{ is associative)} \tag{5.10b}$$

$$a \oplus b = b \oplus a \qquad \text{(}\oplus\text{ is commutative)} \tag{5.10c}$$

$$① \otimes a = a \otimes ① = a \qquad \text{(① is the identity for } \otimes\text{)} \tag{5.10d}$$

$$(a \otimes b) \otimes c = a \otimes (b \otimes c) \qquad \text{(}\otimes\text{ is associative)} \tag{5.10e}$$

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \text{(}\otimes\text{ left distributes over } \oplus\text{)} \tag{5.10f}$$

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \qquad \text{(}\otimes\text{ right distributes over } \oplus\text{)} \tag{5.10g}$$

$$a^{⊛} = ① \oplus (a \otimes a^{⊛}) = ① \oplus (a^{⊛} \otimes a). \qquad \text{(asteration / transitive closure)} \tag{5.10h}$$

When clear from context, we will write $a \otimes b$ as ab.

Star semirings extend **semirings**, which have been used extensively in dynamic programming algorithms, especially in natural language processing (Goodman, 1999; Eisner et al., 2004). Intuitively, addition is used to combine alternate events (in our case alternate paths), and multiplication is used to compose sequences of events that occur together jointly (in our case, the individual arcs along a path). The distributive property enables refactoring, allowing computation to be memoized and reused in a dynamic program. *Star* semirings add the asteration operator, $^{⊛}$, which is used to repeat a sequence zero or more times (in our case, following a loop on a non-simple path). Star semirings have the same syntax as regular expressions, so in some sense they can be thought of as *interpreters* for regular expressions. They are related to **Kleene algebras**, which follow the same laws, plus an additional **idempotency** law:

$$a \oplus a = a, \forall a \in \mathcal{S}. \tag{5.11}$$

We happen to use Kleene algebras in all of our experiments, but idempotency is not strictly required for KGEN.

As an example, the **max-times** semiring on the unit interval, $\mathcal{M} = ([0,1], 0, 1, \max, \times)$, can be made into a star semiring by letting $x^{\circledast} = 1$ for every $x \in [0,1]$.

Furthermore, any star semiring $\mathcal{S} = (\mathcal{A}, ⓪, ①, \oplus, \otimes, {}^{\circledast})$ can be lifted to a star semiring over square matrices in the following way:

$$\mathcal{S}^{d\times d} = (\mathcal{A}^{d\times d}, \mathbf{0}, \mathbf{I}, \oplus, \otimes, {}^{\circledast}), \tag{5.12a}$$

where:

$$\mathbf{0}_{i,j} = ⓪, \tag{5.12b}$$

$$\mathbf{I}_{i,j} = \begin{cases} ①, & \text{if } i = j \\ ⓪, & \text{otherwise,} \end{cases} \tag{5.12c}$$

$$[A \oplus B]_{i,j} = A_{i,j} \oplus B_{i,j}, \tag{5.12d}$$

$$[A \otimes B]_{i,j} = \bigoplus_{k=1}^{n} (A_{i,k} \otimes B_{k,j}), \tag{5.12e}$$

$$A^{\circledast} = \textsc{Kleene}_{\mathcal{S}}(A). \qquad \textit{(Alg. 3)} \tag{5.12f}$$

We describe Kleene's algorithm (Kleene, 1951; O'Connor, 2011) in Algorithm 3. As in the Floyd-Warshall algorithm, $\textsc{Kleene}_{\mathcal{S}}(A)$ is built up iteratively, at each step allowing one more node to be used in paths. Line 5 gives the main recurrence of the dynamic program. A path from i to j will either go through k as an intermediate node ($A^{(k-1)}_{i,k} A^{(k-1)\circledast}_{k,k} A^{(k-1)}_{k,j}$), or not ($A^{(k-1)}_{i,j}$). Some well-known special cases can calculate reachability ($\mathcal{S}$ = Boolean), shortest paths ($\mathcal{S}$ = min-plus),

Algorithm 3: Kleene / Generalized Floyd-Warshall

```
Data: A square matrix $A \in \mathcal{S}^{d\times d}$ with entries from some star semiring $\mathcal{S}$. $A$ can be
      thought of as the edge weights of a directed graph having $\{1,\ldots,d\}$ as vertices.
Result: $A$'s asteration, Kleene$(A)$.
/* $A^{(k)}_{i,j}$ encodes paths from $i$ to $j$ that only use intermediate
   nodes in $\{1,\ldots,k\}$.                                                 */
1 $A^{(0)} \leftarrow A$ ;                     // paths having no intermediate nodes
2 for $k = 1,\ldots,d$ do
3     for $i = 1,\ldots,d$ do
4         for $j = 1,\ldots,d$ do
5             $A^{(k)}_{i,j} \leftarrow (A^{(k-1)}_{i,k} A^{(k-1)\circledast}_{k,k} A^{(k-1)}_{k,j}) \oplus A^{(k-1)}_{i,j}$ ;
6 return Kleene$_{\mathcal{S}}(A) = A^{(n)}$;
```

or can convert an FSA into a regular expression ($\mathcal{S}$ = RegEx). When calculating shortest paths or reachability, $a^{\circledast} = ①$, i.e. loops are never needed, so the $A^{(k-1)\circledast}_{k,k}$ term can be omitted. A straightforward implementation is $O(n^3)$, but in many cases the two inner loops may be vectorized, yielding runtimes closer to $O(n)$. Lifting a star semiring into a square matrix is an effective way in general to construct a star semiring with a *non-commutative* $\otimes$ operation. This is important for our purposes, because we will use $\otimes$ to encode sequences, and we wish to be sensitive to the order of the sequence we are encoding.[4]

5.3.3 Semantic Dependency Parsing Model

Since our focus is on encoding, we return to an arc-independent decoder, as in Logistic-Edge (§2.2.2). Although we found that modeling arc interactions was beneficial, and achieved state-of-the-art at the time (Chapter 3), the current state-of-the-art in semantic dependency parsing (Dozat and Manning, 2018) uses arc-independent decoding, so this is a reasonable simplifying choice. Let $\boldsymbol{x} = x_1,\ldots,x_n$ be an input sentence of length n. Let $\mathcal{L}$ be the set of $K+1$ possible arc labels: DM's original K arc labels, plus the additional label NoEdge, which represents the

[4]$\oplus$, on the other hand, *must* be commutative, by law (Equation 5.10a). This is important for our encoder to respect the natural invariances of graphs. I.e. alternative paths have no order.

absence of an arc. As in LOGISTICEDGE we consider all token index pairs $1 \leq i, j \leq n$ (in both directions), and treat each ordered pair as a multiclass classification problem over $\mathcal{L}$. We do not do any singleton pruning or enforce any constraints on the output, so decoding can be done for each pair independently and in parallel. We do prune pairs where $|i - j| > D$; the maximum arc distance $D = 20$ was chosen so as to prune out fewer than 0.5% of gold arcs. We also prune the label set $\mathcal{L}$ to the most common 20 labels (out of 59), pruning out fewer than 0.5% of gold arcs, but reducing the label set size significantly.

For candidate source index i and destination index j, we calculate a dense representation with a parameterized function $\mathbf{f}_\theta(\boldsymbol{x}, i, j)$ (variants of $\mathbf{f}$ will be described in §5.3.3). Then we pass the arc representation through an MLP, followed by a $\mathrm{softmax}$ to define a distribution over $\mathcal{L}$ (§5.3.3). Parameters are learned to minimize the negative log likelihood of the training data under our model. We use Adam (Kingma and Ba, 2015b) with weight decay and parameter dropout for optimization (see Section A.3 for further optimization details).

Token Representations

We learn a lookup embedding $\mathbf{v}_x \in \mathbb{R}^{d_w}$ for every word that appears in the training or development data. Word vectors are initialized with normalized GloVe 840B embeddings (Pennington et al., 2014a) ($d_w = 100$), and updated during training. Given a sentence $\boldsymbol{x}$, we run a 1-layer bidirectional LSTM over $\boldsymbol{x}$, concatenate the forward and backward hidden state of each token, and apply a learned affine layer to obtain a contextualized token vector $\mathbf{h}_i \in \mathbb{R}^{d_w}$ for each token index $1 \leq i \leq n$.

Arc Representations

We experiment with three alternative ways of incorporating syntax to produce arc representations: none (syntax-unaware), GCN, and KGEN. GCN and KGEN both construct directed graph G based on a preprocessed syntactic dependency parse. The nodes of G are the n tokens of $\boldsymbol{x}$, plus a synthetic node labeled ROOT, representing the syntactic root of the sentence. First we add every arc $i \xrightarrow{\ell} j$

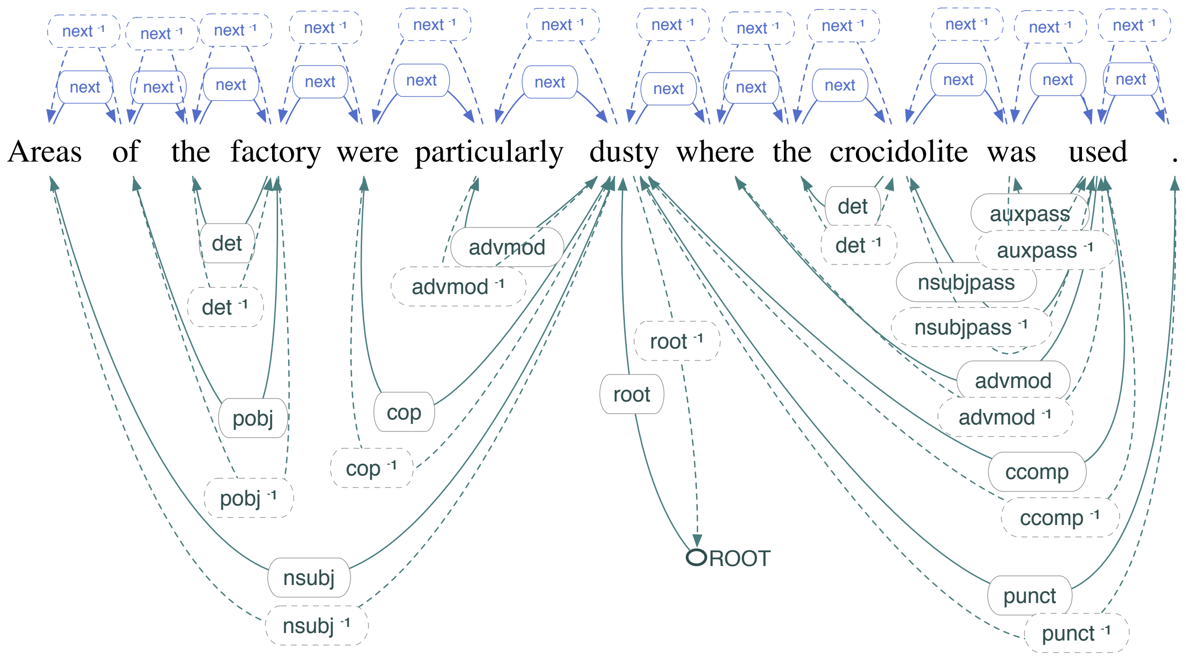


Figure 5.6: Our example sentence, along with the multigraph that will be embedded and contextualized. Solid blue "next" arcs (above the sentence) are added between each node and its successor. Solid green arcs are syntactic dependency arcs (predicted in preprocessing). Every solid arc has a corresponding dashed *backward* arc in the opposite direction.

in the dependency parse to G. Then we add a NEXT arc from every token to its successive token. These arcs allow the graph network access to the linear order of the sentence. For every FORWARD arc $i \xrightarrow{\ell} j$, we a add a BACKWARD arc $j \xrightarrow{\ell^{-1}} i$ to G (see Figure 5.6). For the GCN model, we add a self-loop $i \xrightarrow{\text{SELFLOOP}} i$ to every node i in the graph. In GCNs, this is necessary in order to propagate information from a node to itself in successive GCN layers; in KGEN, this is unnecessary.

Baseline: No Graph Contextualization For our first baseline, we use no graph contextualization step. We form an arc representation for the ordered pair (i, j) by concatenating their respective token representations, along with three distance features.

$$\mathbf{f}(\boldsymbol{x}, i, j) = \mathbf{h}_i; \mathbf{h}_j; [\text{abs}(i - j), \mathbb{1}_{i<j}, \mathbb{1}_{i=j}]. \tag{5.13}$$

Baseline: Graph Convolutional Network For our second baseline, we reimplement the graph convolutional network (GCN) model of Marcheggiani and Titov (2017), which we will refer to as M&T for short. M&T was originally designed to perform dependency-based semantic role labeling on the CoNLL-2009 dataset (Hajič et al., 2009). As a testament to the generality and flexibility of graph-based methods, it can be used virtually without modification for semantic dependency parsing. We follow the model faithfully, and we will restate it here for completeness. GCNs consist of K layers, and in each layer a node is updated based on a function of its adjacent arcs and nodes. Let $\mathbf{u}_i^{(k)}$ denote the representation of the i^{th} node at the k^{th} layer. Each layer is defined inductively in terms of the previous layer, and the current layer's parameters. In M&T, the update function is defined as follows:

$$\mathbf{u}_j^{(0)} = \mathbf{h}_j, \qquad \textit{(base layer)} \quad (5.14a)$$

$$\mathbf{u}_j^{(k+1)} = \text{ReLU}\left(\sum_{i \xrightarrow{\ell} j \in G} g^{(k)}_{i \xrightarrow{\ell} j} \left(\mathbf{V}^{(k)}_{\text{dir}(\ell)} \mathbf{u}_i^{(k)} + \mathbf{b}_\ell^{(k)} \right) \right), \qquad \textit{(successive layer update)} \quad (5.14b)$$

where,

$$g^{(k)}_{i \xrightarrow{\ell} j} = \sigma\left(\hat{\mathbf{v}}^{(k)}_{\text{dir}(\ell)} \cdot \mathbf{u}_i^{(k)} + \hat{b}_\ell^{(k)} \right), \qquad \textit{(edge gate)} \quad (5.14c)$$

and node representations can be read from the final layer:

$$\text{GCN}(G, \{\mathbf{h}_i\}_{i \le n}) = \{\mathbf{u}_i^{(K)}\}_{i \le n}. \qquad \textit{(result)} \quad (5.14d)$$

ReLU is the rectified linear unit (Hahnloser et al., 2000), $\text{ReLU}(x) = \max(0, x)$. The parameter matrices $\mathbf{V}^{(k)}_{\text{dir}(\ell)}$ and vectors $\hat{\mathbf{v}}^{(k)}_{\text{dir}(\ell)}$ are looked up based on the only *direction* of the arc label, $\text{dir}(\ell) \in \{\text{Forward}, \text{Backward}, \text{SelfLoop}\}$, not the label itself. This choice was originally made in order to control the model size (Marcheggiani and Titov, 2017). In the first layer of a GCN, each node's representation contains information from its immediate neighbors. At the K^{th} layer, information has been propagated from all other nodes within K hops. We form an arc representation for the ordered pair (i, j) by concatenating their contextualized token representations, along with three distance features:

$$\mathbf{f}(\boldsymbol{x}, i, j) = \mathbf{u}_i^{(K)}; \mathbf{u}_j^{(K)}; [\text{abs}(i - j), \mathbb{1}_{i<j}, \mathbb{1}_{i=j}]. \tag{5.15}$$

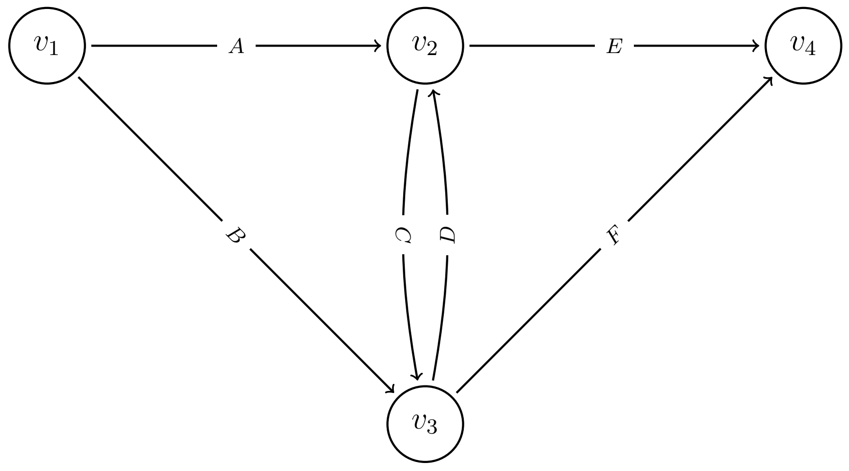
If nodes i and j are within K hops of each other in G, then their representations are "aware" of each other, in a sense. But note that the same vector $\mathbf{u}_i^{(K)}$ is reused for *all* arc representations emanating from the node i; only the distance features are calculated specifically for the pair (i, j). So for the model to be effective, $\mathbf{u}_i^{(K)}$ must encode information about its relationship to *all* other candidate destination nodes.

Our Model: Kleene Graph Encoder Network Here we describe our novel use of Kleene's algorithm as a graph contextualization step. First we will describe it in general, without specializing to a specific star semiring. Then we will describe the specific star semiring, maxmul, that we use in our experiments. maxmul was chosen because of its experimental success as a sequence encoder (Section 5.2), and its straightforward extension into a star semiring.

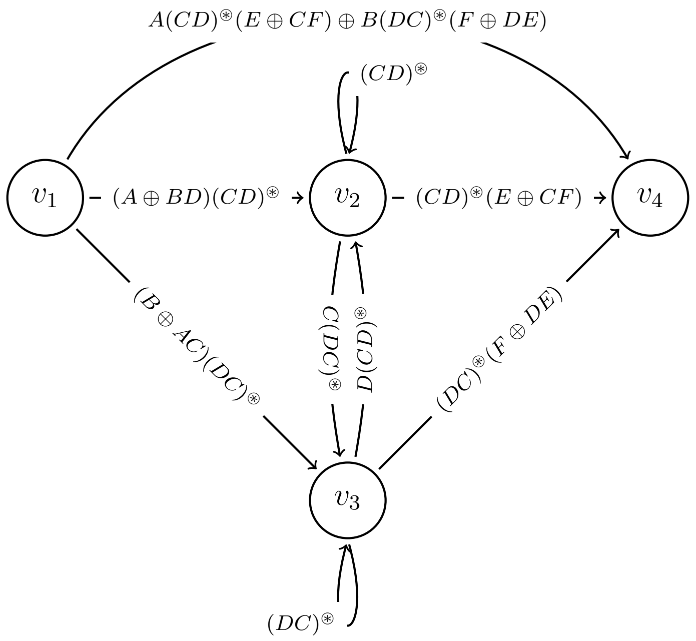
Given the directed multigraph $G = (\{1, \ldots, n+1\}, A_G)$ (as in Figure 5.6), we embed G's arcs into some star semiring $\mathcal{S}$ with a differentiable parameterized function $\mathbf{S} : A_G \to \mathcal{S}$. We create an adjacency matrix $A \in \mathcal{S}^{n \times n}$ where

$$A_{i,j} = \bigoplus_{v_i \xrightarrow{\ell} v_j \in A_G} \mathbf{S}(v_i \xrightarrow{\ell} v_j). \tag{5.16}$$

In other words, if there is no arc from i to j, then $A_{i,j} = ⓪$; if there are multiple arcs, then they are embedded and summed. We then run Kleene's Algorithm (Algorithm 3) to obtain a "contextualized" adjacency matrix $A^{\circledast} = \textsc{Kleene}_{\mathcal{S}}(A)$. See Figure 5.7 for an illustration of this contextualization step. The contextualized arc representation for (i, j) can then be read directly from the matrix, $A^{\circledast}_{i,j}$. When $\mathcal{S}$'s operations are differentiable, then the entire network is differentiable and can be



(a) A directed graph G.



$A(CD)^{\circledast}(E \oplus CF) \oplus B(DC)^{\circledast}(F \oplus DE)$
$(CD)^{\circledast}$
v_1
$(A \oplus BD)(CD)^{\circledast}$
v_2
$(CD)^{\circledast}(E \oplus CF)$
v_4
$(B \oplus AC)(DC)^{\circledast}$
$C(DC)^{\circledast}$
$D(CD)^{\circledast}$
$(DC)^{\circledast}(F \oplus DE)$
v_3
$(DC)^{\circledast}$

(b) Kleene(G).

Figure 5.7: All paths in G.

learned end-to-end. We call the part of the network including $\mathbf{S}$ and KLEENE a Kleene graph encoder network.

The batched maxmul **star semiring.** Here we describe how the **Soft Patterns (SoPa)** model described in Section 5.2 can be extended into a star semiring, and so can be used in the KGEN framework. Let $\mathcal{M}^{d\times d}$ be the **max-times** star semiring lifted to $d \times d$ matrices (Equation 5.12). Concretely, element-wise $\max$ is used for $\oplus$, with the zero matrix as its identity; maxmul (Equation 5.5) is used for $\otimes$, with identity I; and $\textsc{Kleene}_{\mathcal{M}}$ is used for $^{\circledast}$. Such a matrix can be thought of in SoPa terms as the transition matrix for one input for one pattern. We embed arcs into $\mathcal{M}^{d\times d}$ similarly to our calculation of transition matrices in SoPa (Equation 5.3):

$$\left[\mathbf{S}(s \xrightarrow{\ell} t)\right]_{i,j} = \begin{cases} \sigma\left(\mathbf{w}_{i,j} \cdot [\mathbf{h}_s; \mathbf{h}_t; \mathbf{v}_\ell] + a_{i,j}\right), & \text{if } j = i \text{ or } j = i+1 \\ 0, & \text{otherwise,} \end{cases} \tag{5.17}$$

where $\mathbf{w}_{i,j}$ is a vector of parameters, $a_{i,j}$ is a scalar bias parameter, $\mathbf{h}_s$ is the token representation of token s, $\mathbf{v}_\ell$ is a lookup embedding of the arc label ℓ, and σ is the sigmoid function. Like in Equation 5.5, our transition matrix has two non-zero diagonals: the main diagonal, and one above it. Similarly to how SoPa uses an array of a few hundred patterns, we add another index $1 \leq b \leq B$, where B is the number of patterns[5]:

$$\left[\mathbf{S}(s \xrightarrow{\ell} t)\right]_{b,i,j} = \begin{cases} \sigma\left(\mathbf{w}_{b,i,j} \cdot [\mathbf{h}_s; \mathbf{h}_t; \mathbf{v}_\ell] + a_{b,i,j}\right), & \text{if } j = i \text{ or } j = i+1 \\ 0, & \text{otherwise.} \end{cases} \tag{5.18}$$

The star semiring operations extend along this new dimension by operating on each pattern independently.

Once $A^{\circledast}$ has been calculated, we extract the first row of each transition matrix, and concatenate

[5]In our experiments, we use $B = 500$ and $d = 5$ (Appendix A.3). Since the pattern length d is small and fixed, the fact that asteration in $\mathcal{M}^{b\times d\times d}$ is $O(Bd^3)$ still quite reasonable. Our GPU implementation is closer to $O(d)$.

them, along with distance features:

$$\mathbf{f}(\boldsymbol{x}, s, t) = \left[\mathbf{A}^{\circledast}{}_{s,t}\right]_{1,1,:}; \ldots; \left[\mathbf{A}^{\circledast}{}_{s,t}\right]_{B,1,:}; [\text{abs}(s-t), \mathbb{1}_{s<t}, \mathbb{1}_{s=t}]. \tag{5.19}$$

To have exactly matched the SoPa model, we would have taken the single $(1, d)^{\text{th}}$ entry from each pattern, which represents the weight of having matched the entire pattern, from start to end. To avoid the extensive tuning of pattern lengths that SoPa required, we simply take the whole first row, which gives the weight of matching each initial prefix of the pattern.

Our SoPa experiments (§5.2) suggest that $\mathcal{M}^{b\times d\times d}$ is an effective choice for encoding sequences. We also choose $\mathcal{M}$ as a base semiring because its closure can be calculated without worry of overflow, unlike the vanilla sum-product semiring (which overflows $x^{\circledast} = \infty$ for any $x \geq 1$).

Output Layer

As a final step, we pass the computed arc representation for (i, j) through a two-layer multilayer perceptron with output dimension $|\mathcal{L}|$, and take a softmax to get a probability distribution over $\mathcal{L}$:

$$P(\ell \mid, \boldsymbol{x}, i, j; \theta) = [\text{softmax}\,(\text{MLP}(\mathbf{f}_\theta(\boldsymbol{x}, i, j))]_\ell\,. \tag{5.20}$$

5.3.4 Experiments

Dataset

As in Chapters 2 and 3, we test our model on the task of semantic dependency parsing (Oepen et al., 2014, 2015). We use the English portion of the 2015 SDP shared task data (Oepen et al., 2015), and focus on predicting the **DM** formalism. See Section 2.1 for a description of the formalisms and task. For our current preliminary experiments, we have filtered out training data sentences with more than 20 tokens. This leaves 15,428 sentences, out of the original 33,964. We test on all sentences of the in-domain test set.

Baselines

Graph convolutional networks have been recently used to encode syntactic graphs in order to improve (dependency-based) semantic role labeling (M&T; Marcheggiani and Titov, 2017). We test a reimplementation of M&T, retrained on SDP data. SDP graphs are in nearly the same format as the dependency-based SRL graphs used in the original paper (CoNLL-2009; Hajič et al., 2009), so the same model can be retrained and used for semantic dependency parsing without adaptation.

We train and compare models which have the following basic architecture (§5.3.3):

- run a biLSTM over a sentence to get contextualized vectors for each token,
- optionally run a graph contextualization step (GCN or KGEN) that propagates information over predicted syntactic dependency arcs to get syntax-contextualized vectors, and
- for each candidate pair of source and destination tokens, feed their representation into an MLP to get a probability distribution over SDP arc labels (or no label).

To fairly evaluate and compare KGEN, we vary only the method used in the graph contextualization step, keeping the rest of the model fixed. We test a (a) syntax-free model, with no graph contextualization step; (b) GCN; and (c) KGEN. We use a 1-layer GCN in our experiments, as that was found to be the best-performing on the CoNLL-2009 data. We run an ablation test for each of the three models *without* the biLSTM step.

We also compare to two variants of the current state-of-the-art SDP system of Dozat and Manning (D&M; 2018), and two variants of our own previously state-of-the-art system described in Chapter 3 (NeurboParser; Peng et al., 2017a). Our syntax-free baseline (BiLSTM $\rightarrow$ MLP) is most directly comparable to NEURBOPARSER BASIC and D&M BASIC. Both NeurboParser and D&M are syntax-free systems that run on top of a biLSTM and predict each edge (mostly) independently. NeurboParser scores arcs with an MLP and is trained with structured hinge loss, while D&M scores arcs with a biaffine module, following Dozat and Manning (2017). D&M found gains from using a biaffine module instead of an MLP. NEURBOPARSER FREDA3 adds higher-order factors, with

Performance (dev. data, $n \leq 20$)	**P**	**R**	$\mathbf{F}_1$
MLP	75.4	70.2	72.7
GCN $(K = 1) \rightarrow$ MLP	87.5	86.2	86.8
KGEN $\rightarrow$ MLP	91.5	88.8	90.2
BiLSTM $\rightarrow$ MLP	91.0	89.0	90.0
BiLSTM $\rightarrow$ **GCN** $(K = 1) \rightarrow$ MLP	91.5	89.9	$\underline{90.7}$
BiLSTM $\rightarrow$ **KGEN** $\rightarrow$ MLP	90.9	89.5	90.2

Table 5.4: Labeled precision, recall, and F_1 on development data, filtered to sentences of length less than or equal to 20. The upper three rows are models tested *without* any biLSTM. The first row is a syntax-free model, with no graph contextualization step. **GCN** is a reimplementation of the graph convolutional network of Marcheggiani and Titov (2017), and **KGEN** is our own Kleene Graph Encoder Network. The lower three rows add a biLSTM under each of the three tested models.

significant gains in performance, but also significant costs in decoding time and implementation complexity. We opted to keep decoding simple, to focus on the novel contribution. D&M +CHAR +LEMMA adds subword features based on the individual characters in the words' spellings, giving an even more significant gain. In principle, those gains are unrelated to the novelties we have introduced, and are still available to our systems.

5.3.5 Results

We report labeled precision, recall and F_1 on development data for our models and baselines in Table 5.4. The upper section contains models without a biLSTM, and the lower section contains models run with a biLSTM. The straight MLP with no context performs poorly, unsurprisingly; it has access only to token lookup embeddings, and other than the distance features, has no knowledge of the linear order of the sentence. Both GCN and KGEN on the other hand, perform surprisingly well without an LSTM, with KGEN significantly (4.7 absolute F_1) better. These two models are making use of the linear order of the sentence, but only through the input graph G (Figure 5.6). The GCN tested has 1 layer, so a token's representation only has access to a window limited to the previous token, the next token, and any direct neighbors in the syntactic graph. KGEN on the

Performance (test data)	**P**	**R**	**F$_1$**
MLP	70.6	64.2	67.2
GCN $(K=1) \rightarrow$ MLP	82.6	80.0	81.3
KGEN $\rightarrow$ MLP	88.4	83.7	<u>86.0</u>
BiLSTM $\rightarrow$ MLP	85.4	81.4	83.3
BiLSTM $\rightarrow$ **GCN** $(K=1) \rightarrow$ MLP	85.6	82.3	84.0
BiLSTM $\rightarrow$ **KGEN** $\rightarrow$ MLP	86.5	84.5	85.5
NEURBOPARSER BASIC	-	-	89.4
NEURBOPARSER FREDA3	-	-	90.4
D&M BASIC	-	-	91.4
D&M +CHAR +LEMMA	-	-	**93.7**

Table 5.5: Labeled precision, recall, and F$_1$ on the test data. The top three rows are models tested *without* any biLSTM. The first row is a syntax-free model, with no graph contextualization step. **GCN** is a reimplementation of the graph convolutional network of Marcheggiani and Titov (2017), and **KGEN** is our own Kleene Graph Encoder Network. The next three rows add a biLSTM under each of the three tested models. In the lower four rows, we quote performance results from recently published state-of-the-art systems, for reference.

other hand is able to take much better advantage of the linear order information encoded in G. Because G is fully connected and includes both directions for every arc, the output of KGEN is **fully contextualized**, even without a biLSTM. By this we mean that the representation for every token pair (s, t) includes information from *every* arc in G (and hence also every token in $\boldsymbol{x}$).

Both GCN and the syntax-free model have much improved performance when run on top of a biLSTM, with gains of 17.3 and 3.9 absolute F$_1$, respectively. The gains accurately reflect how much more information is available to the model. Somewhat surprisingly, KGEN is unimproved with an LSTM; A reasonable conclusion is that KGEN is capturing the relevant linear order context of the sentence *as well as a biLSTM*. The two variants of KGEN are tied as the best performing models on the development set, beating the BiLSTM$\rightarrow$GCN by 1.4 absolute F$_1$. Of course, the development set was used for tuning, so the results on the unseen test data in the following paragraph should be taken more seriously.

We report the same statistics for the test data in Table 5.5, along with four previously published results for reference. Similar trends are seen on the test data, except that here, KGEN *without* a biLSTM is the best performing model we tested. A possible explanation is that the biLSTM is redundant, so the extra model complexity only leads to overfitting. Scores are roughly 5 points lower than on the development data across the board, likely because we tested on test sentences of all lengths, whereas longer sentences were filtered out of the train and development splits. This, along with our very limited hyperparameter tuning, also likely explains the bulk of the gap between our models and NEURBOPARSER/D&M. More comprehensive experiments could possibly close the gap.

The results are already quite promising though, with KGEN without a biLSTM outperforming a GCN even with a biLSTM. The fact that KGEN is able to achieve strong performance without a biLSTM means that it is able to learn both sequential order and syntactic context from a single unified multigraph. This bodes well for its potential when adding more preprocessed graphs to its input multigraph. There is also potential for its decisions to be interpretable. Assuming the same interpretation methods used in SoPa (§5.2.6) can be applied to KGEN, then KGEN's use of sequential order and syntactic context are both interrogable.

Chapter 6

Conclusion

In this thesis we explored the dual goals of decoding and encoding linguistic graphs, with experiments and motivating applications in natural language processing, especially semantic analysis. These two goals roughly map to the two questions: "How do you automatically produce a graph from text?", and "How do you use such a graph, once you have it?".

In Chapters 2 through 4, we introduced methods for *decoding* linguistic graphs—given natural language text, how to produce a graph representing some aspect of its linguistic structure. In Chapter 2, we experimented with linguistically motivated features and constraints in a linear model for semantic dependency parsing, achieving strong results in a shared task. In Chapter 3, we built on the winning model of that shared task, updating it to use deep neural networks. We further showed how all three semantic dependency parsing formalisms could be modeled and predicted jointly in a multitask learning setup, leading to state-of-the-art performance. In Chapter 4, we showed that jointly predicting the nodes and edges of *connected* graphs is an NP-hard problem, relating it to the prize-collecting Steiner tree problem. We introduced an extension of PCST, NEWCS, which is more suitable for graph prediction. We extended techniques for PCST solving to NEWCS solving, and showed how these techniques could be used for semantic parsing and other related NLP tasks.

In Chapter 5, we introduced new models for *encoding* sequences and graphs. We first introduced a new, highly interpretable model for text encoding called SoPa. SoPa is made up of many small FSAs with neural weights, and we show experimentally that despite its limited form, SoPa is a capable model for encoding sequences. Then we extended SoPa to encode graphs (Section 5.3), making use of the fact that inference in WFSAs follows the semiring laws, and in some cases, the star semiring laws. This allowed us to encode entire graphs using dynamic programming. We tested our new model, the Kleene Graph Encoder Network (KGEN), by encoding a syntactic graph and predicting semantic dependencies. We compared to a strong Graph Convolutional Network baseline, and showed that KGEN resulted in a bigger performance gains on the SDP task.

One question left unexplored in this thesis is: when predicting a graph G, how should one predict G's node set? In our experiments we only predict arcs, given a fixed set of nodes. In fact, we show in Chapter 4 that jointly modeling nodes and arcs is a hard problem, but we do give an approximate decoding algorithm for it (BEASLEYGRAPH, §1). In future work, we hope to use this BEASLEYGRAPH, along with a KGEN for encoding syntax, in a NeurboParser for the abstract meaning representation. This would be a robust demonstration of the power of the graph-to-graph pipeline described in Chapter 1.

While we focused in this thesis on graphs representing syntactic and semantic aspects of sentences, graph-based networks have applications across NLP (*e.g.,* Yasunaga et al., 2017; Cetoli et al., 2017; Bastings et al., 2017), and beyond (*e.g.,* Zhou, 2017; Defferrard et al., 2016; Kipf and Welling, 2016; Zellers et al., 2018).

Appendix A

Implementation Details

A.1 Neural Turbo Semantic Parsing

Each input token is mapped to a concatenation of three real vectors: a pre-trained word vector; a randomly-initialized word vector; and a randomly-initialized POS tag vector.[1] All three are updated during training. We use 100-dimensional `GloVe` (Pennington et al., 2014b) vectors trained over Wikipedia and Gigaword as pre-trained word embeddings. To deal with out-of-vocabulary words, we apply word dropout (Iyyer et al., 2015b) and randomly replace a word w with a special unk-symbol with probability $\frac{\alpha}{1+\#(w)}$, where $\#(w)$ is the count of w in the training set.

Models are trained for up to 30 epochs with Adam (Kingma and Ba, 2015b), with $\beta_1 = \beta_2 = 0.9$, and initial learning rate $\eta_0 = 10^{-3}$. The learning rate η is annealed at a rate of 0.5 every 10 epochs (Dozat and Manning, 2017). We apply early-stopping based on the labeled F_1 score on the development set.[2] We set the maximum number of iterations of AD^3 to 500 and round decisions when it doesn't converge. We clip the ℓ_2 norm of gradients to 1 (Graves, 2013; Sutskever et al., 2014), and we do not use mini-batches. Randomly initialized parameters are sampled from a

[1]There are minor differences in the part-of-speech data provided with the three formalisms. For the basic models, we use the POS tags provided with the respective dataset; for the multitask models, we use the (automatic) POS tags provided with DM.

[2]Micro-averaged labeled F_1 for the multitask models.

Hyperparameter	**Value**
Pre-trained word embedding dimension	100
Randomly-initialized word embedding dimension	25
POS tag embedding dimension	25
Dimensions of representations ϕ and ψ	100
MLP layers	2
BiLSTM layers	2
BiLSTM dimensions	200
Rank of tensor r	100
α for word dropout	0.25

Table A.1: Hyperparameters used in the NeurboParser experiments (§3.1.4,§3.2.4).

uniform distribution over $\left[-\sqrt{6/(d_\text{r}+d_\text{c})}, \sqrt{6/(d_\text{r}+d_\text{c})}\right]$, where d_r and d_c are the number of the rows and columns in the matrix, respectively. An ℓ_2 penalty of $\lambda = 10^{-6}$ is applied to all weights. Other hyperparameters are summarized in Table A.1.

We use the same pruner as Martins and Almeida (2014), where a first-order feature-rich unlabeled pruning model is trained for each task, and arcs with posterior probability below 10^{-4} are discarded. We further prune labeled structures that appear less than 30 times in the training set. In the development set, about 10% of the arcs remain after pruning, with a recall of around 99%.

A.2 Soft Patterns

We implemented all neural models in PyTorch,[3] and the **Hard** baseline in scikit-learn (Pedregosa et al., 2011).[4] We train using Adam (Kingma and Ba, 2015a) with a batch size of 150. We use 300-dimensional GloVe 840B embeddings (Pennington et al., 2014a) normalized to unit length. We randomly initialize all other parameters. Our MLP has two layers. For regularization, we use dropout.[5]

In all cases, we tune the hyperparameters of our model on the development set by running 30 iterations of random search. The full list of hyperparameters explored for each model can be found in Table A.2. Finally, we train all models for 250 epochs, stopping early if development loss does not improve for 30 epochs.

[3]https://pytorch.org/

[4]http://scikit-learn.org/

[5]**DAN** uses word dropout instead of regular dropout as its only learnable parameters are the MLP layer weights.

Type	Values	Models
Patterns	{5:10,4:10,3:10,2:10}, {6:10,5:10,4:10}, {6:10,5:10,4:10,3:10,2:10}, {6:20,5:20,4:10,3:10,2:10}, {7:10,6:10,5:10,4:10,3:10,2:10}	**SoPa**
Learning rate	0.01, 0.05, 0.001, 0.005	**SoPa, DAN, BiLSTM, CNN**
Dropout	0, 0.05, 0.1, 0.2	**SoPa, BiLSTM, CNN**
MLP hid. dim.	10, 25, 50, 100, 300	**SoPa, DAN, BiLSTM, CNN**
Hid. layer dim.	100, 200, 300	**BiLSTM**
Out. layer dim.	50, 100, 200	**CNN**
Window size	4, 5, 6	**CNN**
Word dropout	0.1, 0.2, 0.3, 0.4	**DAN**
Log. reg. param	1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001	**Hard**
Min. pattern freq.	2–10, 0.1%	**Hard**

Table A.2: The hyperparameters explored in our Soft Patterns experiments. **Patterns**: the number of patterns of each length. For example, {5:20,4:10} means 20 patterns of length 5 and 10 patterns of length 4. **MLP hid. dim.**: the dimension of the hidden layer of the MLP. **Hid. layer dim.**: the BiLSTM hidden layer dimension. **Out. layer dim.**: the CNN output layer dimension. **Window size**: the CNN window size. **Log. reg. param**: the logistic regression regularization parameter. **Min. pattern freq.**: minimum frequency for a pattern to be included as a logistic regression feature, expressed either as absolute count or as relative frequency in the train set. **Models**: the models to which each hyperparameter applies (see Section 5.2.4).

Hyperparameter Type	**Value**	**Models**
Word embedding dim.	100	all
Lemma embedding dim.	50	all
POS tag embedding dim.	50	all
LSTM hidden layer dim.	100	all
MLP hidden dim.	100	all
MLP number of layers	2	all
Number of convolutional layers	1	**GCN**
Pattern Length	5	**KGEN**
Number of patterns	500	**KGEN**
Syntax label embedding dim.	50	**KGEN**
Learning rate	10^{-3}	all
Minibatch size	1	all
Gradient clipping	5.0	all
Parameter dropout	0.05	all
Weight decay	10^{-6}	all
Max. train sentence length	20	all

Table A.3: The hyperparameter values used in our KGEN experiments.

A.3 Kleene Graph Encoder Networks

Table A.3 gives the full list of hyperparameter values used in our experiments: Hyperparameter values were not tuned; they were guessed at based on prior work. We trained all models for 200 epochs, stopping early if development loss did not improve for 30 epochs. We implemented all models in PyTorch.[6] Other than word embeddings, all parameters were randomly initialized. We used both weight decay and parameter dropout for regularization.

[6] https://pytorch.org

Bibliography

Mariana S. C. Almeida and André F. T. Martins. 2015. Lisbon: Evaluating TurboSemanticParser on multiple languages and out-of-domain data. In *Proc. of SemEval*. 27, 28, 32, 33

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah Smith. 2016. Many languages, one parser. *TACL* 4:431–444. 35

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR* 6:1817–1853. 35

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*. 74

Collin Baker, Michael Ellsworth, and Katrin Erk. 2007. Semeval'07 task 19: Frame semantic structure extraction. In *Proc. of SemEval*. 4, 55

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *Proc. of ACL*. 2

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proc. of LAW VII & ID*. 3, 9, 37

Yehoshua Bar-Hillel, M Perles, and Eli Shamir. 1961. On formal properties of simple phrase structure grammars. 56

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph

Convolutional Encoders for Syntax-aware Neural Machine Translation. *arXiv:1704.04675 [cs]* ArXiv: 1704.04675. http://arxiv.org/abs/1704.04675. 78, 79, 98

Leonard E. Baum and Ted Petrie. 1966. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics* 37(6):1554–1563. https://doi.org/10.1214/aoms/1177699147. 60

John E. Beasley. 1989. An SST-based algorithm for the Steiner problem in graphs. *Networks* 19(1):1–16. 38, 44

Yoshua Bengio, Rèjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *JMLR* 3:1137–1155. 25

Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David Williamson. 1993. A Note on the Prize Collecting Traveling Salesman Problem. *Mathematical Programming* 59(3):413–420. https://doi.org/10.1007/BF01581256. 39

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP*. 35

Benjamin Bloem-Reddy and Yee Whye Teh. 2019. Probabilistic symmetry and invariant neural networks. *arXiv:1901.06082 [cs, stat]* ArXiv: 1901.06082. http://arxiv.org/abs/1901.06082. 82

Carlo E. Bonferroni. 1936. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commerciali di Firenze* 8:3–62. 27

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-Recurrent Neural Network. In *Proc. of ICLR*. 73

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics* 18(4):467–479. 18

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proc. of*

ACL. http://arxiv.org/abs/1704.07092. 37

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of CoNLL*. 29

Rich Caruana. 1997. Multitask learning. *Machine Learning* 28(1):41–75. 20

A. Cetoli, S. Bragaglia, A. D. O'Harney, and M. Sloan. 2017. Graph Convolutional Networks for Named Entity Recognition. *arXiv:1709.10053 [cs]* ArXiv: 1709.10053. http://arxiv.org/abs/1709.10053. 78, 79, 98

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies*. Association for Computational Linguistics, NAACL '01, pages 1–9. 56

Yining Chen, Sorcha Gilroy, Kevin Knight, and Jonathan May. 2018. Recurrent neural networks as weighted language recognizers. In *Proc. of NAACL*. 73

Kyunghyun Cho. 2015. Natural language understanding with distributed representation. ArXiv:1511.07916. 24

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of EMNLP*. http://www.aclweb.org/anthology/D14-1179. 73

Noam Chomsky. 1959. A note on phrase structure grammars. *Information and Control* 2(4):393–395. https://doi.org/10.1016/S0019-9958(59)80017-6. 3

YJ Chu and TH Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica* 14. 1, 41

Axel Cleeremans, David Servan-Schreiber, and James L McClelland. 1989. Finite state automata and simple recurrent networks. *Neural computation* 1(3):372–381. https://doi.org/10.1162/neco.1989.1.3.372. 73

John Cocke. 1969. *Programming Languages and Their Compilers: Preliminary Notes*. New York University, New York, NY, USA. 1

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. of ICML*. 35

Ann Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proc. of LREC*. 10

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 2005. Minimal recursion semantics: An introduction. *Research on Language & Computation* 3(4):281–332. 3, 37

Fabrizio Costa, Paolo Frasconi, Vincenzo Lombardo, and Giovanni Soda. 2003. Towards Incremental Parsing of Natural Language Using Recursive Neural Networks. *Applied Intelligence* 19:9–25. https://doi.org/10.1023/A:1023860521975. 1

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *TACL* 3:433–447. https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/480. 73

Hal Daumé III. 2007. Frustratingly easy domain adaptation. In *Proc. of ACL*. 29, 35

Dmitry Davidov and Ari Rappoport. 2008. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated SAT analogy questions. In *Proc. of ACL*. 69

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Enhanced sentiment learning using twitter hashtags and smileys. In *Proc. of COLING*. http://www.aclweb.org/anthology/C10-2028. 60

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford dependencies: A crosslinguistic typology. In *Proc. of LREC*. 4

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proc. of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*. Manchester, UK, pages 1–8. 18

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *arXiv:1606.09375 [cs, stat]* ArXiv: 1606.09375. http://arxiv.org/abs/1606.09375. 78, 79, 98

Bonnie Dorr, Nizar Habash, and David Traum. 1998. A thematic hierarchy for efficient generation from lexical-conceptual structure. In David Farwell, Laurie Gerber, and Eduard Hovy, editors, *Machine Translation and the Information Soup: Proceedings of the Third Conference of the Association for Machine Translation in the Americas*, Springer, number 1529 in Lecture Notes in Computer Science, pages 333–343. 37

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proc. of ICLR*. 22, 35, 93, 99

Timothy Dozat and Christopher D. Manning. 2018. Simpler but More Accurate Semantic Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 484–490. http://www.aclweb.org/anthology/P18-2077. 85, 93

Markus Dreyer. 2011. *A Non-parametric Model for the Discovery of Inflectional Paradigms from Plain Text Using Graphical Models over Strings*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD, USA. 73

Manfred Droste and Paul Gastin. 1999. The Kleene–Schützenberger theorem for formal power series in partially commuting variables. *Information and Computation* 153(1):47–80. https://doi.org/10.1006/inco.1999.2799. 66

Yantao Du, Fan Zhang, Weiwei Sun, and Xiaojun Wan. 2014. Peking: Profiling syntactic tree parsing techniques for semantic graph parsing. In *Proc. of SemEval*. 27

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proc. of SemEval*. 27

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12:2121–2159. 12

C. W. Duin and A Volgenant. 1987. Some generalizations of the Steiner problem in graphs. *Networks* 17:353–364. 51, 52, 53

C. W. Duin and A. Volgenant. 1989. Reduction tests for the Steiner problem in graphs. *Networks* 19:549 – 567. 51

Jack Edmonds. 1967. *Optimum branchings*. Journal of Research of the National Bureau of Standards. National Bureau of Standards. http://archive.org/details/jresv71Bn4p233. 1, 41

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of ACL*. https://doi.org/10.3115/1073083.1073085. 59, 61

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). https://doi.org/10.18653/v1/W16-5901. 61

Jason Eisner, Eric Goldlust, and Noah A. Smith. 2004. Dyna: A language for weighted dynamic programming. In *Proc. of ACL*. Association for Computational Linguistics, Barcelona, Spain, pages 218–221. 79, 83

Jason M. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. http://www.aclweb.org/anthology/C96-1058. 1

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive science* 14(2):179–211. https://doi.org/10.1207/s15516709cog1402$_1$. 58

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen

Soderland, Daniel S. Weld, and Alexander Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence* 165(1):91–134. https://doi.org/10.1016/j.artint.2005.03.001. 60

Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized multi–task learning. In *Proc. of KDD*. Seattle, WA, USA, pages 109–117. https://doi.org/10.1145/1014052.1014067. 19

Manaal Faruqui and Chris Dyer. 2014. Improving vector space word representations using multilingual correlation. In *Proc. of EACL*. Gothenburg, Sweden, pages 462–471. 18

Marshall L. Fisher. 2004. The Lagrangian relaxation method for solving integer programming problems. *Management Science* 50(12):pp. 1861–1871. http://www.jstor.org/stable/30046157. 45

Nicholas FitzGerald, Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Semantic role labeling with neural network factors. In *Proc. of EMNLP*. 24

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Cmu at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss. In *Proc. of SemEval*. Association for Computational Linguistics, pages 1202–1206. https://doi.org/10.18653/v1/S16-1186. 37, 48

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the Abstract Meaning Representation. In *Proc. of ACL*. 9, 18, 23, 35, 38, 39, 41, 48

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. DeepBank: a dynamically annotated treebank of the Wall Street Journal. In *Proc. of the Eleventh International Workshop on Treebanks and Linguistic Theories*. Lisbon, Portugal, pages 85–96. 10

Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Communications of the ACM* 5(6):345. https://doi.org/10.1145/367766.368168. 53

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef Van Genabith. 2011. #hardtoparse: POS tagging and parsing the Twitterverse. In *AAAI 2011 Workshop on Analyzing Microtext*. pages 20–25. 56

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. of ICML*. 73

Arthur M Geoffrion. 1974. *Lagrangean relaxation for integer programming*. Springer. 45

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics* 28(3):245–288. 9

C. Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation* 4(3):393–405. https://doi.org/10.1162/neco.1992.4.3.393. 73

Kevin Gimpel and Noah A. Smith. 2010. Softmax-margin training for structured log-linear models. Technical Report CMU-LTI-10-008, Carnegie Mellon University. http://lti.cs.cmu.edu/sites/default/files/research/reports/2010/cmulti10008.pdf. 13

Michel X. Goemans and David P. Williamson. 1992. A general approximation technique for constrained forest problems. In *Proc. of SODA*. pages 307–316. http://dl.acm.org/citation.cfm?id=139404.139468. 5, 37, 39

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *JAIR* 57:345–420. 58, 73

C. Goller and A. Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*. volume 1, pages 347–352 vol.1. https://doi.org/10.1109/ICNN.1996.548916. 1

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics* 25(4):573–605. 61, 79, 83

Alex Graves. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer. 24

Alex Graves. 2013. Generating sequences with recurrent neural networks. ArXiv 1308.0850. 99

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2016. A universal framework for inductive transfer parsing across multi-typed treebanks. In *Proc. of COLING*. 35

Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405:947. https://doi.org/10.1038/35016072. 88

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing Prague Czech-English dependency treebank 2.0. In *Proc. of LREC*. 3, 10

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proc. of CoNLL*. pages 1–18. http://www.aclweb.org/anthology/W09-1201. 88, 93

Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proc. of COLING*. https://doi.org/10.3115/992133.992154. 58, 60

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multi-lingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Computational Linguistics* 39(4):949–998. 33

Karl Moritz Hermann, Dipanjan Das, Jason Weston, and Kuzman Ganchev. 2014. Semantic frame identification with distributed word representations. In *Proc. of ACL*. 24

Lee Hetherington. 2004. The MIT finite-state transducer toolkit for speech and language processing. In *Proc. of INTERSPEECH*. 73

Frank L. Hitchcock. 1927. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematical Physics* 6(1):164–189. 31

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780. 24, 58, 73

Björn Hoffmeister, Georg Heigold, David Rybach, Ralf Schlüter, and Hermann Ney. 2012. WFST enabled solutions to ASR problems: Beyond HMM decoding. *IEEE Transactions on Audio, Speech, and Language Processing* 20:551–564. https://doi.org/10.1109/TASL.2011.2162402. 73

F.K. Hwang, D.S. Richards, and P. Winter. 1992. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier Science. 39

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015a. Deep unordered composition rivals syntactic methods for text classification. In *Proc. of ACL*. http://www.aclweb.org/anthology/P15-1162. 69

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015b. Deep unordered composition rivals syntactic methods for text classification. In *Proc. of ACL*. 99

Richard Johansson. 2013. Training parsers on incompatible treebanks. In *Proc. of NAACL*. 35

Richard Johansson and Pierre Nugues. 2008. Dependency-based semantic role labeling of PropBank. In *Proc. of EMNLP*. Honolulu, HI, USA, pages 69–78. 18

Jenna Kanerva, Juhani Luotolahti, and Filip Ginter. 2015. Turku: Semantic dependency parsing as a sequence classification. In *Proc. of SemEval*. 32

Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, Springer US, The IBM Research Symposia Series, pages 85–103. 39

T. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA. 1

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proc. of EMNLP*. http://www.aclweb.org/anthology/D14-1181. 66, 69

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016a. Character-aware neural language models. In *Proc. of AAAI*. 73

Young-Bum Kim, Karl Stratos, and Ruhi Sarikaya. 2016b. Frustratingly easy neural domain adaptation. In *Proc. of COLING*. 29

Diederik Kingma and Jimmy Ba. 2015a. Adam: A method for stochastic optimization. In *Proc. of ICLR*. 101

Diederik P. Kingma and Jimmy Ba. 2015b. Adam: A method for stochastic optimization. In *Proc. of ICLR*. 86, 99

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL* 4:313–327. 22, 23, 35

Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907 [cs, stat]* http://arxiv.org/abs/1609.02907. 57, 78, 79, 98

Steven C. Kleene. 1951. Representation of events in nerve nets and finite automata. Technical Report RM-704, RAND Corporation. 84

Tamara G. Kolda and Brett W. Bader. 2009. Tensor decompositions and applications. *SIAM Review* 51(3):455–500. 31

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 146–157. https://doi.org/10.18653/v1/P17-1014. 82

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proc. of EMNLP*. 35, 41, 45

Joseph B. Kruskal. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* 7(1):48. https://doi.org/10.2307/2033241. 38

Meghana Kshirsagar, Sam Thomson, Nathan Schneider, Jaime Carbonell, Noah A. Smith, and Chris Dyer. 2015. Frame-semantic role labeling with heterogeneous annotations. In *Proc. of ACL*. 35

Marco Kuhlmann. 2014. Linköping: Cubic-time graph parsing with a simple scoring scheme. In *Proc. of SemEval*. 35

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proc. of EMNLP*. 22, 35

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proc. of EMNLP*. 9

Yann LeCun. 1998. Gradient-based learning applied to document recognition. In *Proc. of the IEEE*. 58, 66

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proc. of EMNLP*. https://www.aclweb.org/anthology/D17-1018. 73

Daniel J. Lehmann. 1977. Algebraic structures for transitive closure. *Theoretical Computer Science* 4(1):59–76. https://doi.org/10.1016/0304-3975(77)90056-1. 57

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding CNNs for text: non-linear, non-consecutive convolutions. In *Proc. of EMNLP*. http://aclweb.org/anthology/D15-1180. 74

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing Neural Predictions. In *Proc. of EMNLP*. https://aclweb.org/anthology/D16-1011. 74

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proc. of ACL*. 30, 31, 35

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding Neural Networks through Representation Erasure. ArXiv:1612.08220. 74

Andy Liaw and Matthew Wiener. 2002. Classification and regression by randomForest. *R News* 2(3):18–22. http://cran.r-project.org/web/packages/randomForest/. 18

Dekang Lin, Shaojun Zhao, Lijuan Qin, and Ming Zhou. 2003. Identifying synonyms among distributionally similar words. In *Proc. of IJCAI*. 60

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming* 45(3):503–528. 12

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward abstractive summarization using semantic representations. In *Proc. of NAACL*. 38, 41

Ivana Ljubić, René Weiskircher, Ulrich Pferschy, Gunnar W. Klau, Petra Mutzel, and Matteo Fischetti. 2006. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. In *Mathematical Progamming, Series B*. 41, 44, 51

Xavier Lluís, Xavier Carreras, and Lluís Màrquez. 2013. Joint arc-factored parsing of syntactic and semantic dependencies. *TACL* 1:219–230. 29, 33

Xuezhe Ma and Eduard Hovy. 2016. Neural probabilistic model for non-projective MST parsing. ArXiv 1701.00874. 35

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv:1703.04826 [cs]* ArXiv: 1703.04826. http://arxiv.org/abs/1703.04826. 58, 78, 79, 88, 93, 94, 95

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics* 19(2):313–330. 1

André F. T. Martins and Mariana S. C. Almeida. 2014. Priberam: A turbo semantic parser with second order features. In *Proc. of SemEval*. 16, 22, 23, 26, 35, 100

André F. T. Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proc. of ACL*. Sofia, Bulgaria, pages 617–622. 19, 35, 41, 45

André F. T. Martins, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proc. of ACL*. 29, 41

André F. T. Martins, Noah A. Smith, Mário A. T. Figueiredo, and Pedro M. Q. Aguiar. 2011. Dual decomposition with many overlapping components. In *EMNLP*. 16, 23, 45

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proc. of RecSys*. https://doi.org/10.1145/2507157.2507163. 68

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proc. of ACL*. 35, 41

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*. Trento, Italy, pages 81–88. 19, 41

Yusuke Miyao. 2006. From linguistic theory to syntactic analysis: Corpus-oriented grammar development and feature forest model. 10

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proc. of ICML*. 25

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics* 23:269–311. http://www.aclweb.org/anthology/J97-2003. 59

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language* 16(1):69–88. https://doi.org/10.1006/csla.2001.0184. 73

Darren Moore, John Dines, Mathew Magimai-Doss, Jithendra Vepa, Octavian Cheng, and Thomas Hain. 2006. Juicer: A weighted finite-state transducer speech decoder. In *MLMI*. 73

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proc. of NAACL*. http://www.aclweb.org/anthology/N16-1098. 68

Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2018. Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs. *arXiv:1811.01900 [cs, stat]* ArXiv: 1811.01900. https://arxiv.org/abs/1811.01900. 82

Thien Huu Nguyen and Ralph Grishman. 2016. Modeling Skip-Grams for Event Detection with Convolutional Neural Networks. In *Proc. of EMNLP*. https://aclweb.org/anthology/D16-1085. 74

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. JMLR.org, New York, NY, USA, ICML'16, pages 2014–2023. http://dl.acm.org/citation.cfm?id=3045390.3045603. 82

Russell O'Connor. 2011. A very general method of computing shortest paths. `http://r6.ca/blog/20110808T035622Z.html`. Accessed: 2018-04-23. 84

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*. 5, 7, 8, 10, 20, 34, 79, 92

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*. 2, 5, 7, 8, 9, 78, 79, 92

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics* 31(1):71–106. 9

Terence Parsons. 1990. *Events in the Semantics of English: A Study in Subatomic Semantics*. MIT Press, Cambridge, MA. 4

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12:2825–2830. 101

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-based dependency parsing. In *Proc. of ACL*. 35

Hao Peng, Sam Thomson, and Noah A. Smith. 2017a. Deep multitask learning for semantic dependency parsing. In *Proc. of ACL*. 20, 93

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017b. Cross-sentence n-ary relation extraction with graph LSTMs. *Transactions of the Association for Computational Linguistics* 5(0):101–115. https://www.transacl.org/ojs/index.php/tacl/article/view/1028. 82

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014a. GloVe: Global vectors for word representation. In *Proc. of EMNLP*. http://www.aclweb.org/anthology/D14-1162. 86, 101

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014b. GloVe: Global vectors for word representation. In *Proc. of EMNLP*. 99

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press. 10

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proc. of NAACL*. http://www.aclweb.org/anthology/N16-1076. 73

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proc. of KDD*. https://doi.org/10.1145/2939672.2939778. 74

Michael Roth and Mirella Lapata. 2016. Neural Semantic Role Labeling with Dependency Path

Embeddings. In *Proc. of ACL*. Association for Computational Linguistics, Berlin, Germany, pages 1192–1202. http://www.aclweb.org/anthology/P16-1113. 81

Bernard Roy. 1959. Transitivité et connexité. *C. R. Acad. Sci. Paris* 249:216–218. 53

Dana Rubinstein, Effi Levi, Roy Schwartz, and Ari Rappoport. 2015. How well do distributional models capture different types of semantic knowledge? In *Proc. of ACL*. 63

Alexander M. Rush, Roi Reichart, Michael Collins, and Amir Globerson. 2012. Improved Parsing and POS Tagging Using Inter-sentence Consistency Constraints. In *In Proc. of EMNLP*. pages 1434–1444. http://dl.acm.org/citation.cfm?id=2390948.2391112. 45

Jacques Sakarovitch. 2009. Rational and recognisable power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 105–174. https://doi.org/10.1007/978-3-642-01492-5$_4$. 66

Nathan Schneider, Emily Danchik, Chris Dyer, and Noah A. Smith. 2014. Discriminative lexical semantic segmentation with gaps: running the MWE gamut. *Transactions of the Association for Computational Linguistics* 2:193–206. 56

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11):2673–2681. 24

M. P. Schützenberger. 1961. On the definition of a family of automata. *Information and Control* 4(2):245–270. https://doi.org/10.1016/S0019-9958(61)80020-X. 66

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proc. of CoNLL*. http://www.aclweb.org/anthology/K15-1026. 60

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2016. Symmetric patterns and coordinations: Fast and enhanced representations of verbs and adjectives. In *Proc. of NAACL*. http://www.aclweb.org/anthology/N16-1060. 74

Roy Schwartz, Maarten Sap, Ioannis Konstas, Li Zilles, Yejin Choi, and Noah A. Smith. 2017. The effect of different writing tasks on linguistic style: A case study of the roc story cloze task. In *Proc.of CoNLL*. http://aclweb.org/anthology/K17-1004. 68

Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. Bridging CNNs, RNNs, and weighted finite-state machines. In *Proc. of ACL*. Forthcoming. 58

A. Segev. 1987. The node-weighted Steiner tree problem. *Networks* 17(1):1–17. https://doi.org/10.1002/net.3230170102. 39

Vered Shwartz, Yoav Goldberg, and Ido Dagan. 2016. Improving hypernymy detection with an integrated path-based and distributional method. In *Proc. of ACL*. http://www.aclweb.org/anthology/P16-1226. 74

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of computer and system sciences* 50(1):132–150. https://doi.org/10.1145/130385.130432. 58, 66

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proc. of LREC)*. 1, 2, 3, 78

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of ICLR Workshop*. 74

David Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proc. of EMNLP*. 29

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*. http://www.aclweb.org/anthology/D13-1170. 1, 68

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proc. of ACL*. 29

Vivek Srikumar and Christopher D Manning. 2014. Learning distributed representations for structured output prediction. In *Proc. of NIPS*. 24

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proc. of CoNLL*. 4

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proc. of NIPS*. 99

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, joint syntactic-semantic parsing with stack LSTMs. In *Proc. of CoNLL*. 33

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 1556–1566. https://doi.org/10.3115/v1/P15-1150. 1

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003. Max-margin Markov networks. In *Proc. of NIPS*. Vancouver, British Columbia, Canada, pages 25–32. 11

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2004. Max-margin Markov networks. In *Advances in Neural Information Processing Systems 16*. 26

Hillel Taub-Tabib, Yoav Goldberg, and Amir Globerson. 2015. Template kernels for dependency parsing. In *Proc. of NAACL*. 35

L. Tesnière. 1959. *Éléments de syntaxe structurale*. Éditions Klinksieck. 3

Kapil Thadani. 2014. Approximation strategies for multi-structure sentence compression. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Baltimore, Maryland, pages 1241–1251. 55

Kapil Thadani and Kathleen McKeown. 2013. Sentence Compression with Joint Structural Inference. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, Sofia, Bulgaria, pages 65–74. 55

Sam Thomson, Brendan O'Connor, Jeffrey Flanigan, David Bamman, Jesse Dodge, Swabha Swayamdipta, Nathan Schneider, Chris Dyer, and Noah A. Smith. 2014. CMU: Arc-factored, discriminative semantic dependency parsing. In *Proc. of SemEval*. 7, 35

Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. 2016. Compositional learning of embeddings for relation paths in knowledge base and text. In *Proc. of ACL*. Association for Computational Linguistics, Berlin, Germany, pages 1434–1444. http://www.aclweb.org/anthology/P16-1136. 81

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proc. of ICML*. Banff, Alberta, Canada, pages 104–111. 11, 13

Oren Tsur, Dmitry Davidov, and Ari Rappoport. 2010. ICWSM–a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. In *Proc. of ICWSM*. 69

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. 2015. Evaluation of word vector representations by subspace alignment. In *Proc. of EMNLP*. http://aclweb.org/anthology/D15-1243. 63

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representa-*

tions. https://openreview.net/forum?id=rJXMpikCZ. 82

Ingmar Visser, Maartje EJ Raijmakers, and Peter CM Molenaar. 2001. Hidden markov model interpretations of neural networks. In *Connectionist Models of Learning, Development and Evolution*, Springer, pages 197–206. https://doi.org/10.1007/978-1-4471-0281-$6_2$0. 73

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13(2):260–269. https://doi.org/10.1109/TIT.1967.1054010. 61

Edward Wagstaff, Fabian B. Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. 2019. On the Limitations of Representing Functions on Sets. *arXiv:1901.09006 [cs, stat]* ArXiv: 1901.09006. http://arxiv.org/abs/1901.09006. 82

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proc. of ACL*. Association for Computational Linguistics, Beijing, China, pages 857–862. http://www.aclweb.org/anthology/P15-2141. 37

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proc. of ACL*. 35

Stephen Warshall. 1962. A theorem on boolean matrices. *J. ACM* 9(1):11–12. 53

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proc. of ICML*. Montreal, Quebec, Canada, pages 1113–1120. https://doi.org/10.1145/1553374.1553516. 14

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Martha Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering* 11(2):207–238. 10

Michihiro Yasunaga, Rui Zhang, Kshitijh Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. 2017. Graph-based neural multi-document summarization. *arXiv:1706.06681 [cs]* http://arxiv.org/abs/1706.06681. 78, 80, 98

Ainur Yessenalina, Yisong Yue, and Claire Cardie. 2010. Multi-level structured models for document-level sentiment classification. In *Proc. of EMNLP*. http://www.aclweb.org/anthology/D10-1102. 74

Wenpeng Yin and Hinrich Schütze. 2015. Multichannel Variable-Size Convolution for Sentence Classification. In *Proc. of CoNLL*. http://www.aclweb.org/anthology/K15-1021. 66, 73, 74

Dani Yogatama, Lingpeng Kong, and Noah A. Smith. 2015. Bayesian optimization of text representations. In *Proc. of EMNLP*. http://aclweb.org/anthology/D15-1251. 68

Dani Yogatama and Noah A. Smith. 2014. Linguistic structured sparsity in text categorization. In *Proc. of ACL*. 74

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. In *Proc. of the ICML Deep Learning Workshop*. 74

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and Control* 10(2):189–208. https://doi.org/10.1016/S0019-9958(67)80007-X. 1

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pages 3391–3401. http://papers.nips.cc/paper/6931-deep-sets.pdf. 82

Vicky Zayats and Mari Ostendorf. 2017. Conversation modeling on Reddit using a graph-structured LSTM. *arXiv:1704.02080 [cs]* ArXiv: 1704.02080. http://arxiv.org/abs/1704.02080. 82

Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Proc. of ECCV*. https://doi.org/10.1007/978-3-319-10590-1$_5$3. 74

Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. 2018. Neural motifs: Scene graph parsing with global context. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 82, 98

Xun Zhang, Yantao Du, Weiwei Sun, and Xiaojun Wan. 2016. Transition-based parsing for deep dependency structures. *Computational Linguistics* 42(3):353–389. 33

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. 2014. Greed is good if randomized: New inference for dependency parsing. In *Proc. of EMNLP*. 29

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proc. of ACL*. 35

Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. In *Proc. of COLING*. http://aclweb.org/anthology/C16-1329. 69

Zhenpeng Zhou. 2017. Graph Convolutional Networks for Molecules. *arXiv:1706.09916 [cs, stat]* ArXiv: 1706.09916. http://arxiv.org/abs/1706.09916. 98