October 2019

Machine Learning Models for Efficient and Robust Natural Language Processing

Emma Strubell

MACHINE LEARNING MODELS FOR EFFICIENT AND ROBUST NATURAL LANGUAGE PROCESSING

A Dissertation Presented

by

EMMA STRUBELL

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2019

College of Information and Computer Sciences

MACHINE LEARNING MODELS FOR EFFICIENT AND ROBUST NATURAL LANGUAGE PROCESSING

A Dissertation Presented

by

EMMA STRUBELL

ACKNOWLEDGEMENTS

I would not have been able to complete this dissertation without support from an enormous group of people, a subset of whom I will attempt to give credit here. I want to extend my deepest gratitude to my advisor Andrew McCallum for giving me a chance to try out this NLP thing, and for supporting me not only in my pursuits as a researcher, but also as an outdoorswoman. I am also very grateful to the rest of my thesis committee: Brendan O'Connor, Joe Pater, Mohit Iyyer, and Yoav Goldberg, for their thoughtful feedback on this work, that I wish I had solicited sooner! Much of this work was completed in collaboration with others, and I would particularly like to thank David Belanger, Luke Vilnis, Daniel Andor, David Weiss, Benjamin Roth and Thomas Kollar for helping me learn how to write a great paper and perform correspondingly careful and interesting research. I also want to thank Katherine Silverstein and Ananya Ganesh for their help not only as co-authors but also for enduring my mentorship as I hone those skills! Though we have not published papers together, I am also grateful for early pair-programming sessions with Jinho Choi and Alexandre Passos, where I learned how to engineer efficient NLP code. And although our work together is not included in this dissertation, I also want to thank Elsa Olivetti for being an awesome collaborator, and for showing me that you can be a super successful female scientist without sacrificing friendliness. I also want to extend my gratitude to mentors from my undergraduate career at UMaine who were indispensable for encouraging me to pursue a PhD, and helping me get into graduate school: David Hiebeler, Sudarshan Chawathe, Lawrence Latour and George Markowsky. Finally, I am extremely grateful for the boundless emotional and intellectual support from my best friends, Patrick Verga and Sara Merrick-Albano.

ABSTRACT

MACHINE LEARNING MODELS FOR EFFICIENT AND ROBUST NATURAL LANGUAGE PROCESSING

SEPTEMBER 2019

EMMA STRUBELL

B.S., UNIVERSITY OF MAINE

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Natural language processing (NLP) has come of age. For example, semantic role labeling (SRL), which automatically annotates sentences with a labeled graph representing *who* did *what* to *whom*, has in the past ten years seen nearly 40% reduction in error, bringing it to useful accuracy. As a result, a myriad of practitioners now want to deploy NLP systems on billions of documents across many domains. However, state-of-the-art NLP systems are typically not optimized for cross-domain robustness nor computational efficiency. In this dissertation I develop machine learning methods to facilitate fast and robust inference across many common NLP tasks.

First, I describe paired learning and inference algorithms for *dynamic feature selection* which accelerate inference in linear classifiers, the heart of the fastest NLP models, by 5–10 times. I then present *iterated dilated convolutional neural networks*

(ID-CNNs), a distinct combination of network structure, parameter sharing and training procedures that increase inference speed by 14–20 times with accuracy matching bidirectional LSTMs, the most accurate models for NLP sequence labeling. Finally, I describe *linguistically-informed self-attention* (LISA), a neural network model that combines multi-head self-attention with multi-task learning to facilitate improved generalization to new domains. We show that incorporating linguistic structure in this way leads to substantial improvements over the previous state-of-the-art (syntax-free) neural network models for SRL, especially when evaluating out-of-domain. I conclude with a brief discussion of potential future directions stemming from my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

INTRODUCTION

Core natural language processing (NLP) tasks such as part-of-speech tagging, syntactic parsing and entity recognition have come of age thanks to advances in machine learning. For example, the task of *semantic role labeling* (annotating *who* did *what* to *whom*) has seen nearly 40% error reduction over the past decade. NLP has reached a level of maturity long-awaited by domain experts who wish to leverage natural language analysis to inform better decisions and effect social change. By deploying these systems at scale on billions of documents across many domains practitioners can consolidate raw text into structured, actionable data. These cornerstone NLP tasks are also crucial building blocks to higher-level natural language understanding (NLU) that our field has yet to accomplish, such as whole-document understanding and human-level dialog.

In order for NLP to effectively process raw text across many domains, we require models that are both robust to different styles of text and computationally efficient. The success described above has been achieved in those limited domains for which we have expensive annotated data; models that obtain state-of-the-art accuracy in these data-rich settings are typically neither trained nor evaluated for accuracy out-of-domain. Users also have practical concerns about model responsiveness, turnaround time in large-scale analysis, electricity costs, and consequently environmental conservation, but the highest accuracy systems also have high computational demand. As hardware advances, NLP researchers tend to increase model complexity in step.

The goal of my research is to facilitate large-scale, real-world natural language understanding by developing machine learning algorithms for natural language processing. Towards this end, this dissertation presents my work advancing computational efficiency and robustness in NLP. To facilitate computational efficiency I describe new training and inference algorithms cognizant of strengths in the latest tensor processing hardware, and eliminate redundant computation through joint modeling across many tasks. I also show that, with careful conditioning between tasks, modeling many tasks in a single model can improve robustness to new domains.

In Chapter 1 I provide an overview of the cornerstone NLP tasks typically modeled as sequence labeling, and the machine learning models for labeling text with those annotations that are the basis for my work. In Chapter 2 I present *dynamic feature selection*: paired learning and inference algorithms for significantly reducing computation in the linear classifiers at the heart of the fastest NLP models. This is accomplished by partitioning the features into a sequence of templates which are ordered such that high confidence can often be reached using only a small fraction of all features. Parameter estimation is arranged to maximize accuracy and early confidence in this sequence. On typical benchmarking datasets for part-of-speech tagging, named entity recognition, and dependency parsing our technique preserves comparable accuracy to models using all features while reducing run-time by 5-10x. In Chapter 3 I describe Iterated Dilated Convolutional Neural Networks (ID-CNNs), a faster alternative to bidirectional LSTMs, the most accurate NLP sequence labeling models. In comparison to traditional CNNs, ID-CNNs have better capacity for large context and structured prediction. Unlike LSTMs whose sequential processing on sentences of length N requires O(N) time even in the face of GPU parallelism, ID-CNNs permit fixed-depth convolutions to run in parallel across entire documents. They embody a distinct combination of network structure, parameter sharing and training procedures that enable dramatic 14-20x test-time speedups while retaining accuracy comparable to the Bi-LSTM-CRF. Chapter 4 presents Linguistically-Informed Self-Attention (LISA), a neural network model that combines multi-head self-attention with multi-task learning across dependency parsing, part-of-speech tagging, predicate detection and SRL. Unlike previous models which require significant pre-processing to prepare syntactic features, LISA can incorporate syntax using merely raw tokens as input, encoding the sequence only once to simultaneously perform parsing, predicate detection and role labeling for all predicates. Syntax is incorporated through the attention mechanism, by training one of the attention heads to focus on syntactic parents for each token. We show that incorporating linguistic structure in this way leads to substantial improvements over the previous state-of-the-art (syntax-free) neural network models for SRL, especially when evaluating out-of-domain, where LISA obtains nearly 10% reduction in error while also providing speed advantages. In Chapter 5, I identify some limitations of my completed work, and describe a number of promising directions for future work inspired by the findings described herein.

CHAPTER 1
BACKGROUND

This thesis explores machine learning methods for the many different linguistic annotations that can be modeled as *sequence labeling*, assigning labels to the tokens in a sequence, where the sequence is typically a sentence or a document. Even more elaborate output structures such as entity spans and syntax trees can be extracted using sequence labeling techniques. In this chapter I first describe the basis for the sequence labeling machine learning models developed in this work (§1.1), and then I describe how these models are typically used to perform the five typical NLP pipeline tasks upon which I evaluate the models in this work: part-of-speech tagging, named entity recognition, syntactic dependency parsing, predicate detection and semantic role labeling (§1.2).

1.1 Machine learning models for sequence labeling in NLP

1.1.1 Conditional probability models for sequence labeling

Let $x = [x_1, \ldots, x_T]$ be our length-T sequence of input tokens and $y = [y_1, \ldots, y_T]$ be per-token output tags. Let D be the domain size of each y_i. We predict the most likely y, given a conditional model $P(y|x)$.

This work considers two factorizations of the conditional distribution. First, we have:

$$P(y|x) = \prod_{t=1}^{T} P(y_t|F(x)), \tag{1.1}$$

where the tags are conditionally independent given some features for x. Given these features, $O(D)$ prediction is simple and parallelizable across the length of the sequence. However, feature extraction may not necessarily be parallelizable. For example, RNN-based features require iterative passes along the length of x; see §1.1.2

We also consider a linear-chain CRF model that couples all of y together:

$$P(y|x) = \frac{1}{Z_x} \prod_{t=1}^{T} \psi_t(y_t|F(x))\psi_p(y_t, y_{t-1}), \tag{1.2}$$

where ψ_t is a local factor, ψ_p is a pairwise factor that scores consecutive tags, and Z_x is the partition function (Lafferty et al., 2001). Prediction in this model requires global search using the $O(D^2T)$ Viterbi algorithm.

CRF prediction explicitly reasons about interactions among neighboring output tags, whereas prediction in the first model compiles this reasoning into the feature extraction step (Liang et al., 2008). The suitability of such compilation depends on the properties and quantity of the data. While CRF prediction requires non-trivial search in output space, it can guarantee that certain output constraints, such as for BIO tagging (Ramshaw and Marcus, 1999), will always be satisfied. It may also have better sample complexity, as it imposes more prior knowledge about the structure of the interactions among the tags (London et al., 2016). However, it has worse computational complexity than independent prediction.

1.1.2 Token representations and feature extraction

One of the most challenging aspects of machine learning is determining the best representation to use for the input. Typically raw inputs, such as an image or the words in a natural language sentence, are mapped to numeric representations suitable as input to a machine learning model via a *feature function*, which we denote F in Equations 3.1 and 3.2 above. The exact parameterization of F is the subject of extensive research in NLP, perhaps even more so than in other areas of machine learning research such as computer vision or speech processing. The inputs to machine learning models for vision and audio processing much more closely mirror the inputs to the brain, whereas in NLP we take words as input, whose biological representations are far less understood (Embicka and Poeppel, 2015; Hubel and Wiesel, 1962; Kaas et al., 1999). As a result, while researchers in other areas can draw upon a great deal of knowledge from neuroscience and signal processing, in developing artificial models for language understanding it is more challenging to take inspiration from what is known about biological systems. An additional challenge in NLP is the inherently discrete nature of text, compared to image and audio data where processing is inherently amenable to continuous representations. Since the most efficient optimization techniques operate in continuous space, in NLP we typically first map discrete representations of language (words and characters) to continuous representations. Developing computationally efficient and robust parameterizations for F is the focus of much of this dissertation. In this section I describe many of the most typical parameterizations of F, upon which the work in this thesis builds. In the first subsection, I describe the sparse, lexicalized feature functions that drove the first wave of statistical models for NLP. The remaining sections explore more recent, highly data-driven approaches which use neural networks to build up rich features based on word similarity and context.

1.1.2.1 Sparse, lexicalized feature functions

The most straightforward representation for a token as input to a classifier is a one-hot vector indicating the word form for that token, typically normalized by e.g. removing casing, limiting to a fixed vocabulary, or more advanced morphological analysis. Indeed, in the first wave of statistical machine learning models for NLP, variants of this type of feature mapping, often referred to as *lexicalized features*, make up the vast majority of feature representations. Common lexicalized features include word form, affixes, capitalization patterns, discrete word cluster membership (Brown et al., 1992), and annotations from other models, such as part-of-speech tags or word lemmas. These features are typically extracted not just for the current word, but also for adjacent words and combined to create n-gram features. Typical models for tagging and parsing consist of 50-100 of these different features mapping to hundreds of thousands to millions of binary features, resulting in a large, sparse binary representation for each token.

Once computed, these sparse feature representations are typically provided to a *linear model*, the most basic classifier for sequence labeling. Given an input $x \in \mathcal{X}$, a set of labels $\mathcal{Y}$, a feature map $\Phi(x, y)$, and a weight vector $\mathbf{w}$, a linear model predicts the highest-scoring label

$$y^* = \arg\max_{y \in \mathcal{Y}} \; \mathbf{w} \cdot \Phi(x, y). \tag{1.3}$$

The parameter $\mathbf{w}$ is usually learned by minimizing a regularized (R) sum of loss functions (ℓ) over the training examples indexed by i

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i \ell(x_i, y_i, \mathbf{w}) + R(\mathbf{w}).$$

Sparse lexicalized features are typically partitioned into a set of *feature templates*, so that the weights, feature function, and dot product factor as

$$\mathbf{w} \cdot \Phi(x, y) = \sum_j \mathbf{w}_j \cdot \Phi_j(x, y) \tag{1.4}$$

for some set of feature templates $\{\Phi_j(x, y)\}$. A feature template corresponds to a set of binary lexicalized features extracted by a single rule. For example, the feature template LEN3-SUFFIX might correspond to all observed length-3 suffices in the training data for a given corpus. For a given token, a given feature template often maps to a one-hot vector (i.e. a given token has exactly one length-3 suffix), but this is not always the case (e.g. templates representing a token's membership in a lexicon). To avoid over-fitting, the range of templates like this is often limited such that only features observed more than some cutoff value (typically a single digit, depending on the size of the data) are considered.

1.1.2.2 Self-supervised pre-training and dense token embeddings

A recent and significant advance in NLP sequence labeling is self-supervised pre-training of dense vector token representations. *Self-supervised* refers to the practice of using token co-occurrences found in natural language text as a readily available training signal for learning token representations. This approach is based on the distributional hypothesis proposed by Harris (1951), which asserts that words with similar meaning are used in similar contexts. There are two main machine learning approaches to pre-training in this way: First, token representations may be *context-independent*, with a single representation learned for each item in the token vocabulary, shared across the entire corpus. When each token corresponds to a word these representations are known as *word embeddings*, a term and approach coined by Bengio et al. (2003), who used these representations as input to a neural network language model. Self-supervised pre-training of these word representations for downstream supervised tasks was proposed by Collobert et al. (2011) in their seminal multi-task neural network model for NLP sequence labeling. More recent popular examples of such pre-training of word representations include the word2vec context-bag-of-words (CBOW) and skip-gram with negative sampling (SGNS) training objectives (Mikolov et al., 2013), GloVe (Pennington et al., 2014), and structured word2vec (Ling et al., 2015a). The main differences between these techniques is whether token representations are trained to be predictive of their context tokens (as in Mikolov et al. (2013)), or of co-occurrence counts (as in Pennington et al. (2014)), and whether the ordering of context tokens is taken into account (Ling et al., 2015a). It has been shown that predict-based models (the SGNS objective in particular) is essentially performing implicit factorization of a word count co-occurrence matrix (Levy and Goldberg, 2014b). There have been many more specialized variants of embeddings trained for better performance on specific tasks, such as word representations that take into account syntactic (Levy and Goldberg, 2014a) or semantic (Rothe and Schütze, 2015) information.

More recently, advances in hardware and methodology have enabled improved pre-training of *context-dependent* token representations, where a given token embedding is a function of its exact context in a sentence or document. These models are typically trained using a language-modeling objective similar to previous approaches, except that token representations are not shared across each instance in the corpus, but instead vary depending on the specific sentence or document containing a token. These models typically employ multiple neural network layers to build a token representation that effectively incorporates multiple aspects of its context, and the representations at each layer for a given token can be combined in different ways, often via task-dependent parameters that are fine-tuned in a secondary training step. The first of these models to achieve great success by demonstrating large accuracy gains across many NLP sequence labeling tasks is the ELMo model (Peters et al., 2018), which consists of multiple LSTM neural network layers (§1.1.2.4), where the first layer builds token representations from character embeddings. Subsequent models include BERT (Devlin et al., 2019), which uses the Transformer neural network architecture (§1.1.2.5) and a different language modeling training objective, GPT and GPT-2 (Radford et al., 2019), which are also Transformer-based with different training. In contrast with the word embedding models described in the previous section and the ELMo model which uses character embeddings, BERT and other models use *wordpiece embeddings* (Schuster and Nakajima, 2012; Wu et al., 2016), which find a middle ground between word and character embeddings, which compared to word embeddings have been found to improve generalization (by removing out-of-domain words and capturing morphology), require a reduced memory footprint, but require increased computation to obtain word embeddings and consequently result in slower inference. A fixed vocabulary of wordpieces are obtained by computing the most common word subsequences in a training corpus, such that the fixed set of wordpieces can be combined to form any word, but the number of wordpieces required to create a word on average is minimized. While this work does contain experiments incorporating ELMo embeddings and therefore character embeddings, since the focus of this work is the morphologically-poor English language all other experiments in this document use pre-trained word embeddings.

1.1.2.3 Feed-forward and convolutional neural networks

The most straightforward neural network extension of the basic linear models described in §1.1.2.1 is the *feed-forward neural network*. Feed-forward networks simply add additional layers of processing to the input before classification via multiple linear transformations, each followed by a *nonlinearity*,[1] a non-linear function applied output. For example, a two-layer feed-forward network applied to an input x_t with transformations $W_c^{(1)}$ and $W_c^{(2)}$ and nonlinearity function σ producing the output representation c_t:

$$c_t = \sigma(W_c^{(2)}\sigma(W_c^{(1)}x_t)), \tag{1.5}$$

Common nonlinearities include tanh, sigmoid, and the rectified linear (ReLU) functions. The linear transformations W_c may increase the dimensionality of the input, or decrease it. Typically the dimension of these transformations is treated as a model hyperparameter.

[1]Without the nonlinearity, multiple applications of linear layers are functionally equivalent to a single linear layer.

Convolutional neural networks (CNNs) in NLP are a straightforward extension of feed-forward networks which simply apply the linear transformation to a window of adjacent tokens, rather than a single token on the input. In NLP convolutions are typically one-dimensional, applied to a sequence of vectors representing tokens rather than to a two-dimensional grid of vectors representing e.g. pixels. In this setting, a convolutional neural network layer is equivalent to applying a linear transformation to a sliding window of width r tokens on either side of each token in the sequence. Here, and throughout the chapter, we do not explicitly write the bias terms in linear transformations. The sliding-window representation c_t for each token x_t is:

$$c_t = W_c \left[x_{t-r}; \ldots; x_{t-1}; x_t; x_{t+1}; \ldots; x_{t+r}\right] \tag{1.6}$$

where $[\cdot;\cdot]$ denotes vector concatenation. Often, multiple convolutional layers are stacked, with the output representation from one layer being provided as input to the next layer, in order to build up richer representations incorporating information from a wider context. As with feed-forward networks, a nonlinearity must be applied after each convolutional layer. In a stacked CNN, the effective size of the context observed by a token at layer k with input radius r is given by $2kr + 1$, i.e. the width grows by $2r$ at each layer.

In NLP CNNs are most commonly applied to (1) character-level modeling (composing characters into token representations to capture morphological information and allow for classification of unseen words) and (2) sentence-level classification, rather than token-level classification. This is because, like linear and feed-forward models, even when stacked as described above they are limited to a fixed-width window of context. In the next sections I will describe neural network models that can incorporate context from an entire sequence into the representation for each token. These models are typically less computationally efficient than linear, feed-forward and convolutional models, but obtain higher accuracy.

1.1.2.4 Recurrent neural networks

Recurrent neural networks incorporate context from the entire sequence by processing tokens incrementally in order, maintaining a state vector h_t that incorporates information at each timestep of the input:

$$h_t = RNN(x_t, h_{t-1}) = W_h\,[x_t; h_{t-1}] \tag{1.7}$$

where $[\cdot;\cdot]$ denotes again vector concatenation. For sequence labeling, it has been shown that incorporating both right- and left-context in a *bidirectional* model is beneficial (Schuster and Paliwal, 1997). Bidirectional models combine RNNs in the forward and backward direction by concatenating their hidden representations at each timestep:

$$h_t = [\overleftarrow{RNN}(x_t, \overleftarrow{h}_{t+1}); \overrightarrow{RNN}(x_t, \overrightarrow{h}_{t-1})] \tag{1.8}$$

In practice, higher accuracy for challenging tasks can be obtained by stacking multiple RNNs, where subsequent RNNs take as input at each timestep the hidden states from the previous RNN, rather than the raw input representations x_t.

Vanilla RNNs, that is RNNs that use just a single linear transformation to integrate information from the current timestep with the hidden state, have been repeatedly shown to effectively model time-dependent data, such as natural language text. However, as the sequence length increases, long term dependencies can be lost due to the vanishing gradient problem (Hochreiter, 1998). The Long Short Term Memory (LSTM) variant of RNNs was designed to overcome this shortcoming (Hochreiter and Schmidhuber, 1997). The LSTM incorporates three additional matrices inside the recurrence which act as a gating mechanism, selectively allowing information flow from previous timesteps as a function of the current timestep. The gates f_t, i_t and

o_t are defined as functions of the input x_t and previous hidden representation h_{t-1} as follows:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{1.9}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{1.10}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{1.11}$$

matrices W and U and bias vectors b are all model parameters learned through supervised training. σ_g denotes the sigmoid activation function, which normalizes the outputs to have values in $[0, 1]$. These representations are combined to produce the hidden representation h_t as follows:

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{1.12}$$

$$h_t = o_t \circ \sigma_h(c_t) \tag{1.13}$$

where $\circ$ denotes element-wise vector multiplication. Activations σ_c and σ_h use the tanh function to normalize outputs into the range $[-1, 1]$. The LSTM out-performs out-perform the vanilla RNN as a token encoder for various NLP tasks, and as a result has become the de facto token encoder in NLP.

There exist many other variants of recurrent neural network architectures with varying internal structure for combining context state and input such as GRUs (Cho et al., 2014), Skip RNNs (Campos et al., 2018), highway LSTMs (Zhang et al., 2015b) or stacked LSTMs with different patterns of connectivity (Zhou and Xu, 2015a). The work in this document implements only the most common LSTM architecture, so the reader is referred to the original works for detailed discussion of these alternatives.

1.1.2.5 Self-attention and Transformer networks

Self-attention is a new alternative to recurrent architectures for incorporating information from the entire sequence into each token representation. Unlike recurrent networks, self-attention can process all tokens in a sequence at once in parallel, and are thus very efficient on modern tensor processing hardware such as GPUs and TPUs. This contextual information is incorporated through a neural *attention mechanism*, essentially a context-dependent weighted average, where the term *self-attention* simply indicates that the weighted average for a given token is over the same sentence as the token is located. In this work, experiments focus on the Transformer instantiation of self-attention (Vaswani et al., 2017). Rather than incrementally building up context from each token in order, the Transformer model uses stacked layers each consisting of multiple self-attention functions to build up rich representations of tokens in context.

More specifically, the Transformer model works as follows. Input token representations x_t are each projected to a representation that is the same size as the output of the self-attention layers. Importantly, a positional encoding vector computed as a deterministic sinusoidal function of t is added to the input, since the self-attention has no innate notion of token position. Alternatively, position embeddings can be used to supply this information to the model, with the downside that position embeddings are limited to a maximum distance determined at train time.

This token representation is provided as input to a series of J residual multi-head self-attention layers with feed-forward connections. Denoting the jth self-attention layer as $T^{(j)}(\cdot)$, the output of that layer $s_t^{(j)}$, and $LN(\cdot)$ layer normalization (Ba et al., 2016), the following recurrence applied to initial input $c_t^{(p)}$:

$$s_t^{(j)} = LN(s_t^{(j-1)} + T^{(j)}(s_t^{(j-1)})) \tag{1.14}$$

gives final token representations $s_t^{(j)}$. Each $T^{(j)}(\cdot)$ consists of: (a) multi-head self-attention and (b) a feed-forward projection.

The multi-head self attention consists of H attention heads, each of which learns a distinct attention function to attend to all of the tokens in the sequence. This self-attention is performed for each token for each head, and the results of the H self-attentions are concatenated to form the final self-attended representation for each token.

Specifically, consider the matrix $S^{(j-1)}$ of T token representations at layer $j-1$. For each attention head h, we project this matrix into distinct key, value and query representations $K_h^{(j)}$, $V_h^{(j)}$ and $Q_h^{(j)}$ of dimensions $T \times d_k$, $T \times d_q$, and $T \times d_v$, respectively. We can then multiply $Q_h^{(j)}$ by $K_h^{(j)}$ to obtain a $T \times T$ matrix of attention weights $A_h^{(j)}$ between each pair of tokens in the sentence. Following Vaswani et al. (2017) we perform scaled dot-product attention: We scale the weights by the inverse square root of their embedding dimension and normalize with the softmax function to produce a distinct distribution for each token over all the tokens in the sentence:

$$A_h^{(j)} = \text{softmax}(d_k^{-0.5} Q_h^{(j)} {K_h^{(j)}}^T) \tag{1.15}$$

These attention weights are then multiplied by $V_h^{(j)}$ for each token to obtain the self-attended token representations $M_h^{(j)}$:

$$M_h^{(j)} = A_h^{(j)} V_h^{(j)} \tag{1.16}$$

Row t of $M_h^{(j)}$, the self-attended representation for token t at layer j, is thus the weighted sum with respect to t (with weights given by $A_h^{(j)}$) over the token representations in $V_h^{(j)}$.

The outputs of all attention heads for each token are concatenated, and this representation is passed to the feed-forward layer, which consists of two linear projections each followed by leaky ReLU activations (Maas et al., 2013). We add the output of the feed-forward to the initial representation and apply layer normalization to give the final output of self-attention layer j, as in Eqn. 1.14.

1.2 Sequence labeling tasks in NLP

The previous section described many of the most common machine learning approaches to sequence labeling in NLP. In this section, I give an overview of many of the most common token-level sequence labeling tasks in NLP: part-of-speech tagging, named entity recognition, syntactic dependency parsing, semantic role labeling and predicate detection. These are all the tasks upon which the models described in this work are evaluated. Although this is far from an exhaustive list of all the annotations studied in NLP and desired by practitioners, these make up a representative suite of the most common tasks included in a typical NLP pipeline alongside (typically deterministic) tokenization and sentence segmentation. The majority of other desired token annotations can be obtained using similar techniques, simply applied to different label sets.

1.2.1 Part-of-speech tagging

Part-of-speech tagging is the task of annotating tokens with their part-of-speech tags, such as noun, verb, adjective, et cetera. There exist distinct sets of tags for many different languages, as well as a set of universal part-of-speech tags which can be used for multilingual modeling (Petrov et al., 2012). Since this work is focused on English NLP, we focus experiments on the annotations in the Wall Street Journal portion of the Penn TreeBank corpus (Marcus et al., 1993b), a collection of 2,499 news articles (about 1 million tokens) from the 1989 Wall Street Journal newspaper annotated with Treebank II style syntax trees (Bies et al., 1995).

This corpus defines 49 part-of-speech tags for English, differentiating between verb tenses, singular and plural nouns, and some types of punctuation.[2] The dataset is divided into 24 sections. In the typical evaluation setup for part-of-speech tagging, sections 0–18 are used for training, sections 19–21 for development, and sections 22–24 as test data. Each token is annotated with exactly one part-of-speech tag, and models are typically evaluated in terms of per-token and sometimes per-sentence accuracy. Modern models typically achieve accuracy between 97–98%, approximately the limit of human annotators for this task (Manning, 2011). Automatic part-of-speech annotations can be obtained as part of syntactic constituency parsing, or labels can be assigned to tokens by simple multi-class classification for each token, which is the approach we take in this work. Typically part-of-speech tagging is the first step after tokenization and sentence segmentation in an English NLP pipeline.

The earliest machine learning models for part-of-speech tagging which can obtain accuracy close to today's state-of-the-art are generative hidden Markov models (HMMs) (Charniak et al., 1993; Church, 1989). More recently part-of-speech tagging models have trended towards discriminative logisic regression (log-linear) models, also known as maximum entropy or MaxEnt models (Ratnaparkhi, 1996; Toutanova et al., 2003). These are still among the most common part-of-speech tagging models found in off-the-shelf NLP tools. More recently, feed-forward (Collobert et al., 2011) and recurrent neural networks have been shown to obtain state-of-the-art performance for part-of-speech tagging (Ling et al., 2015b), though for English the accuracy obtained by these more complex models over simpler linear models is often negligible.

[2]See the annotation guidelines for more details: `https://catalog.ldc.upenn.edu/docs/LDC99T42/tagguid1.pdf`

1.2.2 Syntactic dependency parsing

The next step in typical NLP pipelines following part-of-speech tagging is syntactic parsing, which induces a tree structure over the tokens indicating syntactic relations in the sentence. In this work we focus on *syntactic dependency parsing*, where this tree is made up of edges assigned between tokens in the sentence; no additional (internal) nodes e.g. representing phrases are added to the representation. Each token is assigned one parent, which is another token in the sentence, and one token is designated as the root of the sentence, which is to assigned itself as a parent, or a dummy "root" token. Dependency parsing has been increasingly adopted over constituency parsing due to its more favorable run-time efficiency — For a sentence of length N tokens, expected $O(N)$ parsing time for greedy, transition-based projective parsing and $O(N^2)$ for non-projective parsing (Nivre, 2003, 2009), versus $O(N^3)$ for CKY — and simply implemented models, though information is lost when converting from constituency to more simple dependency parse structure. Currently, the most accurate machine learning model for dependency parsing is a constituency parser whose outputs are converted to dependency trees post-hoc (Dyer et al., 2016).

The benchmark dataset for English syntactic parsing is also the Wall Street Journal portion of the Penn TreeBank (see §1.2.1). For dependency parsing the constituency trees are converted to dependencies with a deterministic set of *head rules*, which can be used to identify the head of a phrase, and an associated mapping to labels on the induced dependency edges. The sets of head rules most commonly used in NLP are the Collins or Yamada and Matsumoto head rules (Collins, 1999; Yamada and Matsumoto, 2003), and the head rules for the Stanford Dependencies framework (de Marneffe and Manning, 2008). Currently, Stanford Dependencies v3.3 is the dependency framework of choice for syntactic parsing. The split of the Penn TreeBank used for dependency parsing differs from that used for part-of-speech tagging: sections 2–21 are used for training, section 23 is used for evaluation, and sections 22 and, more recently in response to highly data driven models, 24 are used for development (Weiss et al., 2015). A model for dependency parsing may or may not include labels on the edges, though recently it is typical for dependency models to perform labeled parsing. Dependency parsing is typically evaluated using labeled attachment score (LAS), unlabeled attachment score (UAS), and sometimes label score (LS). Each of these are measures of per-token accuracy. Labeled attachment score counts a token as correct only if its head and label are assigned correctly. Unlabeled attachment score considers only whether a token's head was assigned correctly, and label score considers only whether the label on the edge is correct. In English, punctuation tokens[3] are typically excluded from evaluation due to inconsistencies in the Penn TreeBank data on where punctuation tokens should attach.

There are two main algorithms for obtaining dependency parses of sentences: *graph-based* dependency parsing and *transition-based* dependency parsing. We describe these two techniques below. Both are used in different models described in this work.

1.2.2.1 Graph-based dependency parsing

Graph-based approaches to dependency parsing directly predict edges between tokens in the syntactic dependency tree. The most common approach to graph-based dependency parsing with neural networks is the *first-order edge-factored* model, which produces representations for the potential edge between each pair of tokens in the sentence, then uses those representations to predict exactly one edge going into each token (i.e., each token has exactly one head). Higher-order models have been shown to increase graph-based parsing accuracy over these simpler models (Bohnet, 2010; Carreras, 2007; Koo and Collins, 2010), but since the rise of neural network models it

[3]Typically defined as having gold part-of-speech tags from the following set: “, ”, ., :, ,

is yet to be shown that explicitly modeling higher-order dependency graph structure improves accuracy in relatively data-rich settings such as English news text.

Since greedy one-best prediction does not guarantee that the predictions form a well-structured tree[4], typically a post-processing step is performed to induce a tree from the model predictions, such as computing the maximum spanning tree given edge scores output by the model (McDonald et al., 2005). Alternatively, the model can be trained with a structured objective that produces a tree during inference, such as the matrix-tree or Chu-Liu-Edmonds algorithm (Koo et al., 2007) which runs in $O(T^2)$ time for a length T sentence. These approaches produce *non-projective* dependency trees, or trees that may have necessarily crossing dependencies when drawn on a plane. For highly *projective* languages such as English or Chinese, it may be desirable to constrain the inference algorithm to predict only projective trees; adding such a structural inductive bias can be particularly helpful in low-data settings. Eisner (1996) proposed dynamic programming algorithms which guarantee projective outputs, at the cost of slower $O(T^3)$ runtime efficiency.

1.2.2.2 Transition-based dependency parsing

As opposed to graph-based parsing where dependencies are predicted directly that form a tree, transition-based parsers instead predict a series of transitions, or shift-reduce operations, which incrementally build up a tree by adding edges between tokens and pushing and popping from a stack. This type of parsing is also known as shift-reduce or incremental parsing. Some of the first neural network-based dependency parsers were transition-based parsers (Andor et al., 2016; Chen and Manning, 2014; Weiss et al., 2015), which used feed-forward networks. More recently, these transition-based approaches, which run very fast on CPU hardware, have been eschewed in favor of more accurate graph-based neural network approaches (Clark et al., 2018; Dozat

[4]The output structure could: (1) contain loops or (2) have more or less than exactly one root.

and Manning, 2017; Kiperwasser and Goldberg, 2016), which can be performed more efficiently on GPU hardware.

More specifically, transition-based parsing algorithms typically use a stack and a buffer of tokens, combined with a specific set of transitions. The buffer is initialized to contain the tokens in the sentence, and the stack is initialized to be empty. At a given step during prediction, given (features of) the state of the stack and the buffer, a model predicts the next transition, which is performed, and this is repeated until the stack and buffer are empty. For a given transition system, the gold-standard series of transitions can be deterministically generated from the gold-standard parse tree.

Different transition systems are used for projective and non-projective parsing. The most basic transition system for projective parsing is called the *arc-standard* transition system (Nivre, 2003). This system defines three transitions: LEFT-ARC$_l$, RIGHT-ARC$_l$ and SHIFT. SHIFT pushes the first token on the buffer to the stack. LEFT-ARC$_l$ adds a dependency arc (with label l) from the token at the top of the buffer to the top of the stack, and pops the stack. RIGHT-ARC$_l$ adds a labeled dependency arc from the token at the top of the stack to that at the top of the buffer, pops the stack, and replaces the top of the buffer with the token that was at the top of the stack.

Another popular transition system for projective parsing is the *arc-eager* transition system, which is designed to produce more plausible incremental (partial) trees by adding both left and right edges as soon as the head and dependent tokens are available, as opposed to in the arc-standard system, where either left- or right-dependents (typically right) are not attached to heads until each token has found all of its dependents (Nivre, 2004). In other words, the arc-standard system performs purely bottom-up parsing, while the arc-eager system combines bottom-up and top-down parsing. The arc-eager transition system adds a REDUCE transition, which simply pops the stack, and uses a modified RIGHT-ARC transition, which makes the top of the stack the head of the token at the top of the buffer, then pushes the token at the top of the buffer to the stack.

This description focuses on projective transition systems since the focus of this work is English, in which the vast majority of sentences are projective.[5] There also exist a number of transition systems for producing non-projective parses, most notably Nivre's list-based (Nivre, 2008), and swap-based approaches (Nivre, 2009).

1.2.3 Named entity recognition

Named entity recognition is the task of identifying typed spans of tokens corresponding to named entities, such as *Washington* and *United States of America*. The two most commonly used benchmark datasets for English named entity recognition are the CoNLL 2003 shared task (Sang and Meulder, 2003) and the CoNLL 2012 (Pradhan et al., 2012) subset[6] of the OntoNotes corpus v5.0 (Hovy et al., 2006; Pradhan et al., 2006). Entities in the CoNLL 2003 corpus are labeled with one of four types: PER, ORG, LOC or MISC, with a fairly even distribution over the four entity types. OntoNotes contains a larger and more diverse set of 19 different entity types, adding: ORDINAL, PRODUCT, NORP, WORK_OF_ART, LANGUAGE, MONEY, PERCENT, CARDINAL, GPE, TIME, DATE, FACILITY, LAW, EVENT and QUANTITY. The OntoNotes corpus also covers a wider range of text genres, including telephone conversations, web text, broadcast news and translated documents, whereas the CoNLL-2003 text covers only newswire. The sizes of the two corpora in terms of documents, sentences, tokens and entities are given in Table 1.1.

Named entity recognition is often modeled similarly to part-of-speech tagging, where each token is assigned a label, with non-entity tokens receiving the O label.

[5]Chen and Manning (2014) report that 99.9% of syntax trees converted to Stanford dependencies in the Penn TreeBank training data are projective.

[6]The CoNLL 2012 split includes pivot text, which is typically excluded from the data for training and evaluating named entity recognition models.

Dataset		Train	Dev	Test
CoNLL-2003	Tokens	204,567	51,578	46,666
	Sentences	14,041	3,250	3,453
	Documents	945	215	230
	Entities	23,499	5,942	5,648
OntoNotes 5.0	Tokens	1,088,503	147,724	152,728
	Sentences	59,924	8,528	8,262
	Documents	2,483	319	322
	Entities	81,828	11,066	11,257

Table 1.1: Statistics of NER datasets

Typically a boundary encoding is used to assist in classification, i.e. the label of the first token in an entity span is pre-pended with B- and remaining labels in that span pre-pended with I; this is known as BIO or IOB encoding (Ramshaw and Marcus, 1999). Ratinov and Roth (2009) report that, for non-structured models, a slightly more elaborate BILOU (also known as IOBES) encoding, which additionally distinguishes between the final token (L) in a span, and single-token spans (U) out-performs the more basic BIO encoding scheme.

Named entity recognition is typically evaluated using F1 score, which is the harmonic mean of precision and recall. Precision measures, of the spans that were predicted, how many were correct, and is calculated as the number of true positives divided by the total number of predicted spans. Recall measures how many of the correct spans were predicted, and is computed as the number of true positives divided by the total number of spans in the gold standard data. Precision and recall are computed at the segment level, rather than at the token level, so all the tokens in a span need to be identified correctly in order for that span to be considered correct (a true positive).

LSTMs (Hochreiter and Schmidhuber, 1997) were used for NER as early as the CoNLL shared task in 2003 (Hammerton, 2003), but due to lack of hardware, like many other neural network approaches at the time the model was out-performed by approaches using graphical models with lexicalized features based on word type, affixes and capitalization, and generous use of lexicons (Luo et al., 2015; McCallum and Li, 2003; Passos et al., 2014). (Durrett and Klein, 2014) leverages the parallel co-reference annotation available in the OntoNotes corpus to predict named entities jointly with entity linking and co-reference.

More recently, a wide variety of neural network architectures for NER have been proposed. Collobert et al. (2011) employ a one-layer CNN with pre-trained word embeddings, capitalization and lexicon features, and CRF-based prediction. Huang et al. (2015) achieved state-of-the-art accuracy on part-of-speech, chunking and NER using a Bi-LSTM-CRF. Lample et al. (2016) proposed two models which incorporated Bi-LSTM-composed character embeddings alongside words: a Bi-LSTM-CRF, and a greedy stack LSTM which uses a simple shift-reduce grammar to compose words into labeled entities. Their Bi-LSTM-CRF obtained the state-of-the-art on four languages without word shape or lexicon features. Ma and Hovy (2016) use CNNs rather than LSTMs to compose characters in a Bi-LSTM-CRF, achieving state-of-the-art performance on part-of-speech tagging and CoNLL NER without lexicons. Chiu and Nichols (2016) evaluate a similar network but propose a novel method for encoding lexicon matches, presenting results on CoNLL and OntoNotes NER. Yang et al. (2016) use GRU-CRFs with GRU-composed character embeddings of words to train a single network on many tasks and languages.

There have been some approaches showing that incorporating document-level context can help named entity recognition. Ratinov and Roth (2009) incorporate document context in their greedy model by adding features based on tagged entities within a large, fixed window of tokens. Bunescu and Mooney (2004) pose an undirected graphical model over all entities in a document, Sutton and McCallum (2004) employ a skip-chain CRF to model factors between entities while supporting efficient inference, and Finkel et al. (2005) enforce non-local constraints during inference in a CRF.

1.2.4 Semantic role labeling

Semantic role labeling (SRL) is often described as identifying *who* did *what* to *whom*. More precisely, given the (most often verbal) predicates in a sentence, semantic role labeling is the task of, for each predicate, identifying the typed spans of text corresponding to the arguments participating in that predicate's semantic frame.

The CoNLL-2005 shared task (Carreras and Màrquez, 2005) spearheaded machine learning approaches to SRL by providing a version of the Wall Street Journal portion of the Penn TreeBank corpus annotated with predicate-argument structure in the style of PropBank (Palmer et al., 2005). PropBank labels arguments with six predicate-specific labels ARG0–ARG5, known as *core arguments* as well as 18 types of modifiers, e.g. ARGM-MNR, ARGM-NEG. More details can be found in the PropBank annotation guidelines (Bonial et al., 2010). This dataset is one of the main benchmark datasets for SRL, alongside the CoNLL 2009 shared task data (Hajič et al., 2009), which also uses PropBank predicate-argument structure, but annotates only the syntactic heads of phrases, rather than entire spans of text, and the CoNLL 2012 shared task, which includes the annotations from CoNLL 2005 but adds a number of documents from different domains in the OntoNotes corpus, such as broadcast news, telephone conversations and discussion forums.

Since far more annotations are available using PropBank frames, the majority of NLP research has focused on that formalism. Other popular and practical formalisms include VerbNet (Kipper-Schuler, 2005), which annotates arguments with descriptive labels that are shared across all predicates, such as THEME, AGENT and INSTRUMENT. FrameNet (Baker et al., 1998) similarly annotates arguments with labels that are shared across predicates, but FrameNet labels are far more fine-grained than VerbNet (e.g. MOVER, VEHICLE), making it challenging to train a machine learning model for tagging. All of these frameworks extend the verb classes defined by Levin (1993).

There are two main approaches to semantic role labeling: Most recent models identify argument spans for a predicate using token-level tagging with BIO-encoded labels, similar to named entity recognition (He et al., 2017; Marcheggiani et al., 2017; Tan et al., 2018; Zhou and Xu, 2015b). Tags can be predicted independently, or more often decoding can be performed using the Viterbi algorithm. Rather than tagging tokens, another approach is to perform span-level tagging, by considering all possible spans within a sentence up to a maximum length, building feature representations for each span, then assigning labels directly to spans (He et al., 2018; Ouchi et al., 2018). In both cases, SRL is typically evaluated using span-level F1, like named entity recognition.

In NLP pipelines, SRL typically comes after syntactic parsing, following biologically-inspired intuitions that syntactic processing occurs before semantics. Early machine learning approaches to SRL (Johansson and Nugues, 2008; Pradhan et al., 2005; Surdeanu et al., 2007; Toutanova et al., 2008) focused on developing rich sets of linguistic features as input to a linear model, often combined with complex constrained inference e.g. with an ILP (Punyakanok et al., 2008). Täckström et al. (2015) showed that constraints could be enforced more efficiently using a clever dynamic program for exact inference. While most of these techniques require a predicted parse as input, Sutton and McCallum (2005) modeled syntactic parsing and SRL jointly, and Lewis et al. (2015) jointly modeled SRL and CCG parsing.

Collobert et al. (2011) were among the first to use a neural network model for SRL, a CNN over word embeddings combined with globally-normalized inference. However, their model failed to out-perform non-neural models, both with and without multi-task learning with other NLP tagging tasks. FitzGerald et al. (2015) successfully employed neural networks by embedding lexicalized features and providing them as factors in the model of Täckström et al. (2015).

Recently there has been a move away from SRL models which explicitly incorporate syntactic knowledge through features and structured inference. Zhou and Xu (2015b), Marcheggiani et al. (2017) and He et al. (2017) all use variants of deep LSTMs with constrained decoding, while Tan et al. (2018) apply self-attention to obtain state-of-the-art SRL with gold predicates. Unlike most previous work on SRL which unrealistically takes gold predicates as input, He et al. (2017) present end-to-end predicate detection and SRL experiments, predicting predicates using an LSTM, and He et al. (2018) jointly predict SRL spans and predicates in a model based on that of Lee et al. (2017), obtaining state-of-the-art predicted predicate SRL.

Some work has incorporated syntax into neural models for SRL. Roth and Lapata (2016) incorporate syntax by embedding dependency paths, and similarly Marcheggiani and Titov (2017) encode syntax using a graph CNN over a predicted syntax tree, out-performing models without syntax on CoNLL 2009. These works are limited to incorporating partial dependency paths between tokens, and Marcheggiani and Titov (2017) report that their model does not out-perform syntax-free models on out-of-domain data.

1.2.4.1 Predicate detection

A prerequisite for semantic role labeling is *predicate detection*, identifying the predicates in a sentence. Sometimes this also includes predicate sense disambiguation, since different senses of the same predicate lemma often correspond to different semantic frames. Most annotated data for semantic role labeling focuses on verbal predicates (CoNLL 2005, 2009), though CoNLL 2012 also includes annotations of nominal predicates. In much work on semantic role labeling, gold-standard predicates are provided as input to the system, though recent work is moving away from this less realistic setting and instead evaluating *end-to-end* predicate detection and semantic role labeling performance. This is similar to how English syntactic dependency parsing is more typically evaluated using automatic, rather than gold-standard, part-of-speech tags as input.

Predicate detection models are typically trained as taggers similar to named entity recognition using BIO-encoded labels; a small number of predicates (e.g. particle verbs such as *throw out*) are made up of multiple tokens. Like SRL, predicate detection is typically evaluated using the F1 metric.

CHAPTER 2

LEARNING DYNAMIC FEATURE SELECTION FOR FAST SEQUENTIAL PREDICTION

In this chapter I present paired learning and inference algorithms for significantly reducing computation and increasing speed of the vector dot products in the classifiers that are at the heart of many NLP components. This is accomplished by partitioning the features into a sequence of templates which are ordered such that high confidence can often be reached using only a small fraction of all features. Parameter estimation is arranged to maximize accuracy and early confidence in this sequence. Our approach is simpler and better suited to NLP than other related cascade methods. We present experiments in left-to-right part-of-speech tagging, named entity recognition, and transition-based dependency parsing. On the typical benchmarking datasets we can preserve POS tagging accuracy above 97% and parsing LAS above 88.5% both with over a five-fold reduction in run-time, and NER F1 above 88 with more than 2x increase in speed.

2.1 Introduction

This chapter describes a paired learning and inference approach for significantly reducing computation and increasing speed while preserving accuracy in the linear classifiers typically used in many NLP tasks. The heart of the prediction computation in these models is a dot-product between a dense parameter vector and a sparse feature vector. The bottleneck in these models is then often a combination of feature extraction and numerical operations, each of which scale linearly in the size of the feature vector. Feature extraction can be even more expensive than the dot products, involving, for example, walking sub-graphs, lexicon lookup, string concatenation and string hashing. We note, however, that in many cases not all of these features are necessary for accurate prediction. For example, in part-of-speech tagging if we see the word "the," there is no need to perform a large dot product or many string operations; we can accurately label the word a DETERMINER using the word identity feature alone. In other cases two features are sufficient: when we see the word "hits" preceded by a CARDINAL (e.g. "two hits") we can be confident that it is a NOUN.

We present a simple yet novel approach to improve processing speed by dynamically determining on a per-instance basis how many features are necessary for a high-confidence prediction. Our features are divided into a set of *feature templates*, such as current-token or previous-tag in the case of POS tagging. At training time, we determine an ordering on the templates such that we can approximate model scores at test time by incrementally calculating the dot product in template ordering. We then use a running confidence estimate for the label prediction to determine how many terms of the sum to compute for a given instance, and predict once confidence reaches a certain threshold.

In similar work, cascades of increasingly complex and high-recall models have been used for both structured and unstructured prediction. Viola and Jones (2001) use a cascade of boosted models to perform face detection. Weiss and Taskar (2010) add increasingly higher-order dependencies to a graphical model while filtering the output domain to maintain tractable inference. While most traditional cascades pass instances down to layers with increasingly higher recall, we use a single model and accumulate the scores from each additional template until a label is predicted with sufficient confidence, in a stagewise approximation of the full model score. Our technique applies to any linear classifier-based model over feature templates without changing the model structure or decreasing prediction speed.

Most similarly to our work, Weiss and Taskar (2013) improve performance for several structured vision tasks by dynamically selecting features at runtime. However, they use a reinforcement learning approach whose computational tradeoffs are better suited to vision problems with expensive features. Obtaining a speedup on tasks with comparatively cheap features, such as part-of-speech tagging or transition-based parsing, requires an approach with less overhead. In fact, the most attractive aspect of our approach is that it speeds up methods that are already among the fastest in NLP.

We apply our method to left-to-right part-of-speech tagging in which we achieve accuracy above 97% on the Penn Treebank WSJ corpus while running more than five times faster than our 97.2% baseline. We also achieve a five-fold increase in transition-based dependency parsing on the WSJ corpus while achieving an LAS just 1.5% lower than our 90.3% baseline. Named entity recognition also shows significant speed increases. We further demonstrate that our method can be tuned for $2.5 - 3.5$x multiplicative speedups with nearly no loss in accuracy.

2.2 Classification and Structured Prediction

Our algorithm speeds up prediction for multiclass classification problems where the label set can be tractably enumerated and scored, and the per-class scores of input features decompose as a sum over multiple feature templates. Frequently, classification problems in NLP are solved through the use of linear classifiers, which compute scores for input-label pairs using a dot product. These meet our additive scoring criteria, and our acceleration methods are directly applicable.

However, in this work we are interested in speeding up *structured prediction* problems, specifically part-of-speech (POS) tagging and dependency parsing. We apply our classification algorithms to these problems by reducing them to *sequential prediction* (Daumé III et al., 2009). For POS tagging, we describe a sentence's part of speech annotation by the left-to-right sequence of tagging decisions for individual tokens (Giménez and Màrquez, 2004). Similarly, we implement our parser with a classifier that generates a sequence of shift-reduce parsing transitions (Nivre, 2009).

The use of sequential prediction to solve these problems and others has a long history in practice as well as theory. Searn (Daumé III et al., 2009) and DAgger (Ross et al., 2011a) are two popular principled frameworks for reducing sequential prediction to classification by learning a classifier on additional synthetic training data. However, as we do in our experiments, practitioners often see good results by training on the gold standard labels with an off-the-shelf classification algorithm, as though classifying IID data (Bengtson and Roth, 2008; Choi and Palmer, 2012).

Classifier-based approaches to structured prediction are faster than dynamic programming since they consider only a subset of candidate output structures in a greedy manner. For example, the Stanford CoreNLP classifier-based part-of-speech tagger provides a 6.5x speed advantage over their dynamic programming-based model, with little reduction in accuracy. Because our methods are designed for the greedy sequential prediction regime, we can provide further speed increases to the fastest inference methods in NLP.

2.3 Linear models

Our base classifier for sequential prediction tasks will be a *linear model.* Given an input $x \in \mathcal{X}$, a set of labels $\mathcal{Y}$, a feature map $\Phi(x, y)$, and a weight vector $\mathbf{w}$, a linear model predicts the highest-scoring label

$$y^* = \operatorname*{arg\,max}_{y \in \mathcal{Y}} \ \mathbf{w} \cdot \Phi(x, y). \tag{2.1}$$

The parameter $\mathbf{w}$ is usually learned by minimizing a regularized (R) sum of loss functions (ℓ) over the training examples indexed by i

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i \ell(x_i, y_i, \mathbf{w}) + R(\mathbf{w}).$$

In this work, we partition the features into a set of *feature templates*, so that the weights, feature function, and dot product factor as

$$\mathbf{w} \cdot \Phi(x, y) = \sum_j \mathbf{w}_j \cdot \Phi_j(x, y) \tag{2.2}$$

for some set of *feature templates* $\{\Phi_j(x, y)\}$.

Our goal is to approximate the dot products in 2.1 sufficiently for purposes of prediction, while using as few terms of the sum in 2.2 as possible.

2.4 Method

We accomplish this goal by developing paired learning and inference procedures for feature-templated classifiers that optimize both accuracy and inference speed, using a process of *dynamic feature selection*. Since many decisions are easy to make in the presence of strongly predictive features, we would like our model to use fewer templates when it is more confident. For a fixed, learned ordering of feature templates, we build up a vector of class scores incrementally over each prefix of the sequence of templates, which we call the *prefix scores*. Once we reach a stopping criterion based on class confidence (margin), we stop computing prefix scores, and predict the current highest scoring class. Our aim is to train each prefix to be as good a classifier as possible without the following templates, minimizing the number of templates needed for accurate predictions.

Given this method for performing fast inference on an ordered set of feature templates, it remains to choose the ordering. In §2.4.5, we develop several methods for picking template orderings, based on ideas from group sparsity (Swirszcz et al., 2009; Yuan and Lin, 2006), and other techniques for feature subset-selection (Kohavi and John, 1997).

2.4.1 Definitions

Given a model that computes scores additively over template-specific scoring functions as in (2.2), parameters $\mathbf{w}$, and an observation $x \in X$, we can define the i'th *prefix score* for label $y \in \mathcal{Y}$ as:

$$P_{i,y}(x, \mathbf{w}) = \sum_{j=1}^{i} \mathbf{w}_j \cdot \Phi_j(x, y), \tag{2.3}$$

or $P_{i,y}$ when the choice of observations and weights is clear from context. Abusing notation we also refer to the vector containing all i'th prefix scores for observation x associated to each label in $\mathcal{Y}$ as $P_i(x, \mathbf{w})$, or P_i when this is unambiguous.

Given a parameter $m > 0$, called the *margin*, we define a function h on prefix scores:

$$h(P_i, y) = \max\{0, \max_{y' \neq y} P_{i,y'} - P_{i,y} + m\} \tag{2.4}$$

This is the familiar structured hinge loss function as in structured support vector machines (Tsochantaridis et al., 2004), which has a minimum at 0 if and only if class y is ranked ahead of all other classes by at least m.

Using this notation, the condition that some label y be ranked first by a margin can be written as $h(P_i, y) = 0$, and the condition that any class be ranked first by a margin can be written as $\max_{y'} h(P_i, y') = 0$.

2.4.2 Inference

As described in Algorithm 1, at test time we compute prefixes until some label is ranked ahead of all other labels with a margin m, then predict with that label. At train time, we predict until the correct label is ranked ahead with margin m, and return the whole set of prefixes for use by the learning algorithm. If no prefix scores have a margin, then we predict with the final prefix score involving all the feature templates.

Algorithm 1 Inference

```
**Input:** template parameters $\{\mathbf{w}_i\}_{i=1}^k$, margin $m$
and optional (for train time) true label $y$
**Initialize:** $i = 1$
**while** $l > 0 \wedge i \leq k$ **do**
    $l = \max_{y'} h(P_i, y')$ (test) or $h(P_i, y)$ (train)
    $i \leftarrow i + 1$
**end while**
**return** $\{P_j\}_{j=1}^i$ (train) or $\max_{y'} P_{i,y'}$ (test)
```

Algorithm 2 Parameter Learning

```
**Input:** examples $\{(x_i, y_i)\}_i^N$, margin $m$
**Initialize:** parameters $\mathbf{w}_0 = 0$, $i = 1$
**while** $i \leq N$ **do**
    prefixes $\leftarrow$ Infer$(x_i, y_i, \mathbf{w}_i, m)$
    $g_i \leftarrow$ ComputeGradient(prefixes)
    $\mathbf{w}_{i+1} \leftarrow$ UpdateParameters$(\mathbf{w}_i, g_i)$
    $i \leftarrow i + 1$
**end while**
**return** $\mathbf{w}_N$
```

2.4.3 Learning

We split learning into two subproblems: first, given an ordered sequence of feature templates and our inference procedure, we wish to learn parameters that optimize accuracy while using as few of those templates as possible. Second, given a method for training feature templated classifiers, we want to learn an ordering of templates that optimizes accuracy.

We wish to optimize several different objectives during learning: template parameters should have strong predictive power on their own, but also work well when combined with the scores from later templates. Additionally, we want to encourage well-calibrated confidence scores that allow us to stop prediction early without significant reduction in generalization ability.

2.4.4 Learning the parameters

To learn parameters that encourage the use of few feature templates, we look at the model as outputting not a single prediction but a sequence of prefix predictions $\{P_i\}$. For each training example, each feature template receives a number of hinge-loss gradients equal to its distance from the index where the margin requirement is finally reached. This is equivalent to treating each prefix as its own model for which we have a hinge loss function, and learning all models simultaneously. Our high-level approach is described in Algorithm 2.

Concretely, for k feature templates we optimize the following structured max-margin objective (with the dependence of P's on $\mathbf{w}$ written explicitly where helpful):

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{(x,y)} \ell(x, y, \mathbf{w}) \tag{2.5}$$

$$\ell(x, y, \mathbf{w}) = \sum_{i=1}^{i_y^*} h(P_i(x, \mathbf{w}), y) \tag{2.6}$$

$$i_y^* = \min_{i \in \{1..k\}} i \quad \text{s.t.} \quad h(P_i, y) = 0 \tag{2.7}$$

The per-example gradient of this objective for weights $\mathbf{w}_j$ corresponding to feature template Φ_j then corresponds to

$$\frac{\partial \ell}{\partial \mathbf{w}_j} = \sum_{i=j}^{i_y^*} \Phi_j(x, y_{\text{loss}}(P_i, y)) - \Phi_j(x, y). \tag{2.8}$$

where we define

$$y_{\text{loss}}(P_i, y) = \arg\max_{y'} P_{i,y'} - m \cdot I(y' = y), \tag{2.9}$$

where I is an indicator function of the label y, used to define loss-augmented inference.

We add an ℓ_2 regularization term to the objective, and tune the margin m and the regularization strength to tradeoff between speed and accuracy. In our experiments, we used a development set to choose a regularizer and margin that reduced test-time speed as much as possible without decreasing accuracy. We then varied the margin for that same model at test time to achieve larger speed gains at the cost of accuracy. In all experiments, the margin with which the model was trained corresponds to the largest margin reported, i.e. that with the highest accuracy.

2.4.5 Learning the template ordering

We examine three approaches to learning the template ordering.

2.4.5.1 Group Lasso and Group Orthogonal Matching Pursuit

The Group Lasso regularizer (Yuan and Lin, 2006) penalizes the sum of ℓ_2-norms of weights of feature templates (different from what is commonly called "ℓ_2" regularization, penalizing squared ℓ_2 norms), $\sum_i c_i \|w_i\|_2$, where c_i is a weight for each template. This regularizer encourages entire groups of weights to be set to 0, whose templates can then be discarded from the model. By varying the strength of the regularizer, we can learn an ordering of the importance of each template for a given model. The included groups for a given regularization strength are nearly always subsets of one another (technical conditions for this to be true are given in Hastie et al. (2007)). The sequence of solutions for varied regularization strength is called the *regularization path*, and by slight abuse of terminology we use this to refer to the induced template ordering.

An alternative and related approach to learning template orderings is based on the Group Orthogonal Matching Pursuit (GOMP) algorithm for generalized linear models (Lozano et al., 2011; Swirszcz et al., 2009), with a few modifications for the setting of high-dimensional, sparse NLP data (described in §2.4.5.2). Orthogonal matching pursuit algorithms are a set of stagewise feature selection techniques similar to forward stagewise regression (Hastie et al., 2007) and LARS (Efron et al., 2004). At each stage, GOMP effectively uses each feature template to perform a linear regression to fit the gradient of the loss function. This attempts to find the correlation of each feature subset with the residual of the model. It then adds the feature template that best fits this gradient, and retrains the model. The main weakness of this method is that it fits the gradient of the training error which can rapidly overfit for sparse, high-dimensional data. Ultimately, we would prefer to use a development set for feature selection.

2.4.5.2 Sparse Regularized Group Orthogonal Matching Pursuit

Group Orthogonal Matching Pursuit (GOMP) picks a stagewise ordering of feature templates to add to a generalized linear model. At each stage, GOMP effectively uses each feature template to perform a linear regression to fit the gradient of the loss function. This attempts to find the correlation of each feature subset with the residual of the model. It then adds the feature template that best fits this gradient, and retrains the model. We adapt this algorithm to the setting of high-dimensional NLP problems by efficiently inverting the covariance matrices of the feature templates, and regularizing the computation of the residual correlation. This results in a scalable feature selection technique for our problem setting, detailed below.

For purposes of exposition, we will break from the notation of §2.3 that combines features of x and y since the algorithm is designed from a linear-algebraic, regression standpoint that considers the design matrix X and the label matrix Y as separate entities. We will call each group of features G_i, its associated design matrix X_i (the feature matrix for that template).

At each step k, we use our selected set of feature templates/groups and compute the gradient of our loss function on the training data set, call it $r^{(k)}$, and then we select a new feature template G_i with corresponding design matrix X_i to add to our selected groups, by finding the index that maximizes:

$$\arg\max_i \mathbf{tr}\big((r^{(k)})^\top X_i(X_i^\top X_i)^{-1}X_i^\top r^{(k)}\big) \tag{2.10}$$

After adding this template, we retrain the model on the selected set of templates and repeat. Note that this appears to be a difficult optimization problem to solve: some of our templates have hundreds of thousands or even millions of features and computing the inverse $(X_i^\top X_i)^{-1}$ could be expensive – however, due to the special structure of our NLP problems, where each feature template contains one-hot features, this covariance matrix is diagonal and hence trivially invertible.

However, we find in practice that due to the large number of features and relatively small number of examples in our NLP models, this picks very-high cardinality feature templates early on that generalize poorly. The reason becomes apparent when we notice that the correlation-finding subroutine, Equation (2.10), is essentially an un-regularized least squares problem, attempting to regress the loss function gradient onto the data matrices for each template. This suggests we should try a form of regularization, by using some template-dependent constant α_i to regularize the inversion of the covariance matrix:

$$\arg\max_i \mathbf{tr}\big((r^{(k)})^\top X_i(X_i^\top X_i + \lambda_i I)^{-1}X_i^\top r^{(k)}\big) \tag{2.11}$$

Heuristically, we pick this regularization parameter to be a low fractional power of the dimension size for each feature template $\lambda_i = \mathbf{col}(X_i)^{\alpha_i}$, where α_i is picked to be in the regime of $[0.25, 0.5]$.

2.4.5.3 Wrapper Method

The *wrapper method* (Kohavi and John, 1997) is a meta-algorithm for feature selection, usually based on a validation set. We employ it in a stagewise approach to learning a sequence of templates. Given an ordering of the initial sub-sequence and a learning procedure, we add each remaining template to our ordering and estimate parameters, selecting as the next template the one that gives the highest increase in development set performance. We begin the procedure with no templates, and repeat the procedure until we have a total ordering over the set of feature templates. When learning the ordering we use the same hyperparameters as will be used during final training.

While simpler than the Lasso and Matching Pursuit approaches, we empirically found this approach to outperform the others, due to the necessity of using a development set to select features for our high-dimensional application areas.

2.5 Related Work

Our work is primarily inspired by previous research on cascades of classifiers; however, it differs significantly by approximating the score of a single linear model—scoring as few of its features as possible to obtain sufficient confidence.

We pose and address the question of whether a single, interacting set of parameters can be learned such that they efficiently both (1) provide high accuracy and (2) good confidence estimates throughout their use in the lengthening prefixes of the feature template sequence. (These two requirements are both incorporated into our novel parameter estimation algorithm.) In contrast, other work (He et al., 2013; Weiss and Taskar, 2013) learns a separate classifier to determine when to add features. Such heavier-weight approaches are unsuitable for our setting, where the core classifier's features and scoring are already so cheap that adding complex decision-making would cause too much computational overhead.

Other previous work on cascades uses a series of increasingly complex models, such as the Viola-Jones face detection cascade of classifiers (Viola and Jones, 2001), which applies boosted trees trained on subsets of features in increasing order of complexity as needed, aiming to reject many sub-image windows early in processing. We allow scores from each layer to directly affect the final prediction, avoiding duplicate incorporation of evidence.

Our work is also related to the field of learning and inference under test-time budget constraints (Grubb and Bagnell, 2012; Trapeznikov and Saligrama, 2013). However, common approaches to this problem also employ auxiliary models to rank which feature to add next, and are generally suited for problems where features are expensive to compute (*e.g* vision) and the extra computation of an auxiliary pruning-decision model is offset by substantial reduction in feature computations (Weiss and Taskar, 2013). Our method uses confidence scores directly from the model, and so requires no additional computation, making it suitable for speeding up classifier-based NLP methods that are already very fast and have relatively cheap features.

Some cascaded approaches strive at each stage to prune the number of possible output structures under consideration, whereas in our case we focus on pruning the input features. For example, Xu et al. (2013) learn a tree of classifiers that sub-divides the set of classes to minimize average test-time cost. Chen et al. (2012) similarly use a linear cascade instead of a tree. Weiss and Taskar (2010) prune output labels in the context of structured prediction through a cascade of increasingly complex models, and Rush and Petrov (2012) successfully apply these structured prediction cascades to the task of graph-based dependency parsing.

In the context of NLP, He et al. (2013) describe a method for dynamic feature template selection at test time in graph-based dependency parsing. Their technique is particular to the parsing task—making a binary decision about whether to lock in edges in the dependency graph at each stage, and enforcing parsing-specific, hard-coded constraints on valid subsequent edges. Furthermore, as described above, they employ an auxiliary model to select features.

He and Eisner (2012) share our goal to speed test time prediction by dynamically selecting features, but they also learn an additional model on top of a fixed base model, rather than using the training objective of the model itself.

While our comparisons above focus on other methods of *dynamic* feature selection, there also exists related work in the field of general (static) feature selection. The most relevant results come from the applications of *group sparsity*, such as the work of Martins et al. (2011) in *Group Lasso* for NLP problems. The Group Lasso regularizer (Yuan and Lin, 2006) sparsifies groups of feature weights (e.g. feature templates), and has been used to speed up test-time prediction by removing entire templates from the model. The key difference between this work and ours is that we select our templates based on the test-time difficulty of the inference problem, while the Group Lasso must do so at train time. In §2.6.4, we compare against Group Lasso and show improvements in accuracy and speed.

Note that non-grouped approaches to selecting sparse feature subsets, such as boosting and ℓ_1 regularization, do not achieve our goal of fast test-time prediction in NLP models, as they would not zero-out entire templates, and still require the computation of a feature for every template for every test instance.

2.6 Experimental Results

We present experiments on three NLP tasks for which greedy sequence labeling has been a successful solution: part-of-speech tagging, transition-based dependency parsing and named entity recognition. In all cases our method achieves multiplicative speedups at test time with little loss in accuracy.

2.6.1 Part-of-speech tagging

We conduct our experiments on classifier-based greedy part-of-speech tagging. Our baseline tagger uses the same features described in Choi and Palmer (2012).

Model/m	**Tok.**	**Unk.**	**Feat.**	**Speed**
Baseline	97.22	88.63	46	1x
Stagewise	96.54	83.63	9.50	2.74
Fixed	89.88	56.25	1	16.16x
Fixed	94.66	60.59	3	9.54x
Fixed	96.16	87.09	5	7.02x
Fixed	96.88	88.81	10	3.82x
Dynamic/15	96.09	83.12	1.92	**10.36x**
Dynamic/35	97.02	88.26	4.33	**5.22x**
Dynamic/45	97.16	88.84	5.87	3.97x
Dynamic/50	**97.21**	88.95	6.89	3.41x

Table 2.1: Comparison of our models using different margins m, with speeds measured relative to the baseline. We train a model as accurate as the baseline while tagging 3.4x tokens/sec, and in another model maintain $> 97\%$ accuracy while tagging 5.2x, and $> 96\%$ accuracy with a speedup of 10.3x.

We evaluate our models on the Penn Treebank WSJ corpus (Marcus et al., 1993b), employing the typical split of sections used for part-of-speech tagging: 0-18 train, 19-21 development, 22-24 test. The parameters of our models are learned using AdaGrad (Duchi et al., 2011) with ℓ_2 regularization via regularized dual averaging (Xiao, 2009), and we used random search on the development set to select hyperparameters.

This baseline model (**baseline**) tags at a rate of approximately 23,000 tokens per second on a 2010 2.1GHz AMD Opteron machine with accuracy comparable to similar taggers (Choi and Palmer, 2012; Giménez and Màrquez, 2004; Toutanova et al., 2003). On the same machine the greedy Stanford CoreNLP left3words part-of-speech tagger also tags at approximately 23,000 tokens per second. Significantly higher absolute speeds for all methods can be attained on more modern machines.

We include additional baselines that divide the features into templates, but train the templates' parameters more simply than our algorithm. The **stagewise** baseline learns the model parameters for each of the templates in order, starting with only one template—once each template has been trained for a fixed number of iterations, that template's parameters are fixed and we add the next one. We also create a separately-

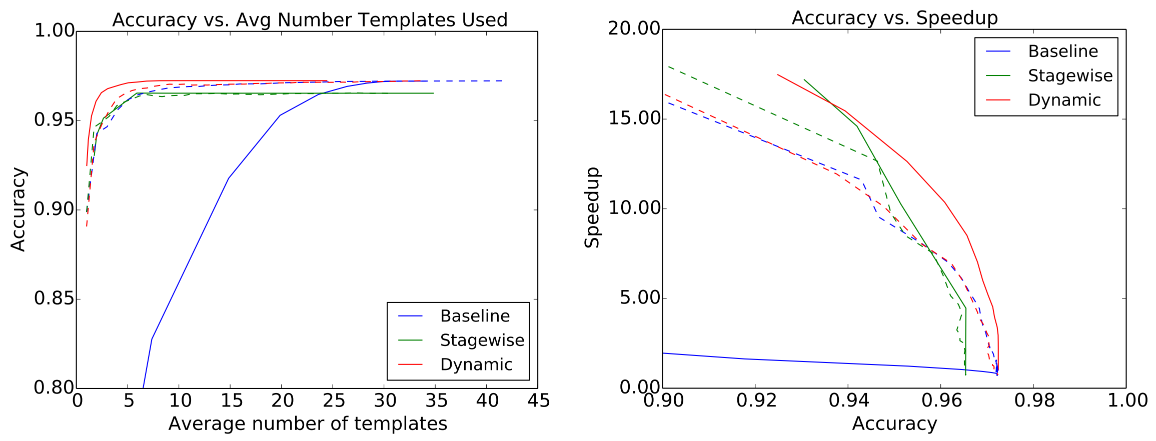

Figure 2.1: Left-hand plot depicts test accuracy as a function of the average number of templates used to predict. Right-hand plot shows speedup as a function of accuracy. Our model consistently achieves higher accuracy while using fewer templates resulting in the best ratio of speed to accuracy.

trained baseline model for each fixed prefix of the feature templates (**fixed**). This shows that our speedups are not simply due to superfluous features in the later templates.

Our main results are shown in Table 2.1. We increase the speed of our baseline POS tagger by a factor of 5.2x without falling below 97% test accuracy. By tuning our training method to more aggressively prune templates, we achieve speed-ups of over 10x while providing accuracy higher than 96%. It is worth noting that the results for our method (**dynamic**) are all obtained from a single trained model (with hyperparameters optimized for $m = 50$, which we observed gave a good speedup with nearly no lossin accuracy on the development set), the only difference being that we varied the margin at test time. Superior results for $m \neq 50$ could likely be obtained by optimizing hyperparameters for the desired margin.

Results show our method (**dynamic**) learns to dynamically select the number of templates, often using only a small fraction. The majority of test tokens can be tagged using only the first few templates: just over 40% use one template, and 75% require at most four templates, while maintaining 97.17% accuracy. On average 6.71 out of

46 templates are used, though a small set of complicated instances never surpass the margin and use all 46 templates. The right hand plot of Figure 2.1 shows speedup vs. accuracy for various settings of the confidence margin m.

The left plot in Figure 2.1 depicts accuracy as a function of the number of templates used at test time. We present results for both varying the number of templates directly (dashed) and margin (solid). The baseline model trained on all templates performs very poorly when using margin-based inference, since its training objective does not learn to predict with only prefixes. When predicting using a fixed subset of templates, we use a different baseline model for each one of the 46 total template prefixes, learned with only those features; we then compare the test accuracy of our dynamic model using template prefix i to the baseline model trained on the fixed prefix i. Our model performs just as well as these separately trained models, demonstrating that our objective learns weights that allow each prefix to act as its own high-quality classifier.

2.6.1.1 Learning the template ordering

As described in §2.4.5, we experimented on part-of-speech tagging with three different algorithms for learning an ordering of feature templates: Group Lasso, Group Orthogonal Matching Pursuit (GOMP), and the wrapper method. For the case of Group Lasso, this corresponds to the experimental setup used when evaluating Group Lasso for NLP in Martins et al. (2011). As detailed in the part-of-speech tagging experiments of Appendix 2.6.4, we found the wrapper method to work best in our dynamic prediction setting. Therefore, we use it in our remaining experiments in parsing and named entity recognition. Essentially, the Group Lasso picks small templates too early in the ordering by penalizing template norms, and GOMP picks large templates too early by overfitting the train error.

2.6.2 Transition-based dependency parsing

We base our parsing experiments on the greedy, non-projective transition-based dependency parser described in Choi and Palmer (2011a). Our model uses a total of 60 feature templates based mainly on the word form, POS tag, lemma and assigned head label of current and previous input and stack tokens, and parses about 300 sentences/second on a modest 2.1GHz AMD Opteron machine.

We train our parser on the English Penn TreeBank, learning the parameters using AdaGrad and the parsing split, training on sections 2–21, testing on section 23 and using section 22 for development and the Stanford dependency framework (de Marneffe and Manning, 2008). POS tags were automatically generated via 10-way jackknifing using the baseline POS model described in the previous section, trained with AdaGrad using ℓ_2 regularization, with parameters tuned on the development set to achieve 97.22 accuracy on WSJ sections 22-24. Lemmas were automatically generated using the ClearNLP morphological analyzer. We measure accuracy using labeled and unlabeled attachment scores excluding punctuation, achieving a labeled score of 90.31 and unlabeled score of 91.83, which are comparable to similar greedy parsers (Choi and Palmer, 2011a; Honnibal and Goldberg, 2013).

Our experimental setup is the same as for part-of-speech tagging. We compare our model (**dynamic**) to both a single **baseline** model trained on all features, and a set of 60 models each trained on a prefix of feature templates. Our experiments vary the margin used during prediction (solid) as well as the number of templates used (dashed).

As in part-of-speech tagging, we observe significant test-time speedups when applying our method of dynamic feature selection to dependency parsing. With a loss of only 0.04 labeled attachment score (LAS), our model produces parses 2.7 times faster than the baseline. As listed in Table 2.2, with a more aggressive margin our

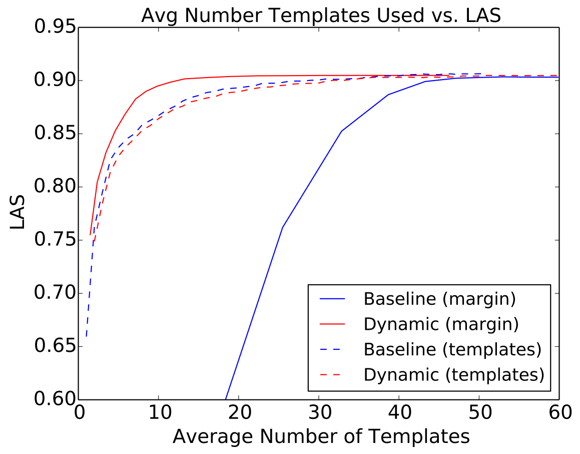

Figure 2.2: Parsing speedup as a function of accuracy. Our model achieves the highest accuracy while using the fewest feature templates.

Model/m	**LAS**	**UAS**	**Feat.**	**Speed**
Baseline	90.31	91.83	60	1x
Fixed	65.99	70.78	1	27.5x
Fixed	86.87	88.81	10	5.51x
Fixed	88.76	90.51	20	2.83x
Fixed	89.04	90.71	30	1.87x
Dynamic/6.5	88.63	90.36	7.81	5.16x
Dynamic/7.1	89.07	90.73	8.57	4.66x
Dynamic/10	90.16	91.70	13.27	3.17x
Dynamic/11	90.27	91.80	15.83	2.71x

Table 2.2: Comparison of our baseline and templated models using varying margins m and numbers of templates.

model can parse more than 3 times faster while remaining above 90% LAS, and more than 5 times faster while maintaining accuracy above 88.5%.

In Figure 2.2 we see not only that our **dynamic** model consistently achieves higher accuracy while using fewer templates, but also that our model (**dynamic**, dashed) performs exactly as well as separate models trained on each prefix of templates (**baseline**, dashed), demonstrating again that our training objective is successful in learning a single model that can predict as well as possible using any prefix of feature templates while successfully selecting which of these prefixes to use on a per-example basis.

Model/m	**Test F1**	**Feat.**	**Speed**
Baseline	88.35	46	1x
Fixed	65.05	1	19.08x
Fixed	85.00	10	2.14x
Fixed	85.81	13	1.87x
Dynamic/3.0	87.62	7.23	2.59x
Dynamic/4.0	88.20	9.45	2.32x
Dynamic/5.0	88.23	12.96	1.96x

Table 2.3: Comparison of our baseline and templated NER models using varying margin m and number of templates.

2.6.3 Named entity recognition

We implement a greedy left-to-right named entity recognizer based on Ratinov and Roth (2009) using a total of 46 feature templates, including surface features such as lemma and capitalization, gazetteer look-ups, and each token's *extended prediction history*, as described in (Ratinov and Roth, 2009). Training, tuning, and evaluation are performed on the CoNLL 2003 English data set with the BILOU encoding to denote label spans.

Our baseline model achieves F1 scores of 88.35 and 93.37 on the test and development sets, respectively, and tags at a rate of approximately 5300 tokens per second on the hardware described in the experiments above. We achieve a 2.3x speedup while maintaining F1 score above 88 on the test set.

2.6.4 Experiments: Learning Template Ordering

As described in §2.4.5, we experimented with 3 different algorithms for learning an ordering of feature templates: Group lasso (**Lasso**), which prunes feature templates based on their norm after ℓ_1 regularization; Group orthogonal matching pursuit (**GOMP**), which selects features by iteratively training a model using the existing features then adds the template that maximizes Eqn. 2.11; and the wrapper method (**wrapper**). We use the methods of setting regularization parameters for

Model	**Templates**	**Accuracy**	**Speed**
Baseline	46	97.22	1x
Lasso	6	90.53	10.97x
Lasso	10	94.33	7.67x
Lasso	15	94.84	5.23x
Lasso	23	96.19	3.31x
GOMP	6	92.18	6.83x
GOMP	10	94.15	4.29x
GOMP	15	96.46	2.83x
GOMP	23	96.81	1.96x
Wrapper	6	96.45	6.13x
Wrapper	10	96.88	3.82x
Wrapper	15	97.01	2.62x
Wrapper	23	97.15	1.70x

Table 2.4: Comparison of the wrapper method for learning template orderings with group lasso and group orthogonal matching pursuit. Speeds are measured relative to the baseline, which is about 23,000 tokens/second on a 2.1GHz AMD Opteron machine.

Lasso and GOMP discussed in §2.4.5. Note that for the case of Group Lasso, this corresponds to the experimental setup used when evaluating Group Lasso for NLP in Martins et al. (2011).

We compare the different methods for picking template orderings in Table 2.4, training and testing models for WSJ POS tagging. To keep concerns separate, we use standard sequential prediction using all templates rather than our dynamic prediction method. We found the wrapper method to be the most successful towards our goal of achieving high accuracy while using as few templates as possible, which is important when using dynamic prediction. While the wrapper method comes within 0.15% of state-of-the-art accuracy using the first 23 out of 46 total templates, neither GOMP nor Lasso are able to exceed 97% accuracy with so few templates.

In terms of speed, Lasso outperforms both GOMP and the wrapper method, predicting 3 times as fast with 23 templates as the baseline, whereas GOMP predicts twice as fast, and the wrapper method 1.7 times baseline speed. It is initially puzzling that for a given number of templates, each model does not achieve the same speed increase. Since each template has only a single active feature for a given test instance, we are computing the same number of multiplications and additions for each model. However, the different models are selecting very different feature templates, whose features can take a widely different amount of time to compute. For example, creating and hashing the string representing a trigram conjunction into a sparse vector takes much longer than creating a feature whose template has only two possible values, *true* and *false*.

This behavior is a reflection of some interesting properties of the template selection methods, which help to explain the superior performance of the wrapper method.

Since the Group Lasso penalizes the norms of the templates as a surrogate for the sparsity (a 0-1 loss), it is naturally biased against large templates and will always include very small templates – the early stages of the regularization path will contain these small templates whose features can be quickly computed. Another likely source of speedup arises from the impact on cache locality of using much smaller cardinality weight vectors. This also explains the poor performance of the induced template ordering. Including small templates early on runs at cross-purposes to our goal of placing highly predictive templates up front for our dynamic prediction algorithm.

In contrast, the wrapper method and GOMP both pick larger, more predictive templates early on since they attempt to minimize loss functions that are unrelated to template size. While GOMP produces better orderings than lasso, in this case the wrapper method works better because the template ordering produced by GOMP is unable to incorporate signal from a validation set and over-fits the training set.

We find that the wrapper method works well with our dynamic training objective, which benefits from predictive, though expensive, features early on in the ordering. When using dynamic prediction, the speed per template used is offset by using significantly fewer templates on examples that are easier to classify.

CHAPTER 3

FAST AND ACCURATE ENTITY RECOGNITION WITH ITERATED DILATED CONVOLUTIONS

Recent advances in GPU hardware have led to the emergence of bi-directional LSTMs as a standard method for obtaining per-token vector representations serving as input to labeling tasks such as NER (often followed by prediction in a linear-chain CRF). Though expressive and accurate, these models fail to fully exploit GPU parallelism, limiting their computational efficiency. This chapter proposes a faster alternative to Bi-LSTMs for NER: Iterated Dilated Convolutional Neural Networks (ID-CNNs), which have better capacity than traditional CNNs for large context and structured prediction. Unlike LSTMs whose sequential processing on sentences of length N requires $O(N)$ time even in the face of parallelism, ID-CNNs permit fixed-depth convolutions to run in parallel across entire documents. We describe a distinct combination of network structure, parameter sharing and training procedures that enable dramatic 14-20x test-time speedups while retaining accuracy comparable to the Bi-LSTM-CRF. Moreover, ID-CNNs trained to aggregate context from the entire document are more accurate than Bi-LSTM-CRFs while attaining 8x faster test time speeds.

3.1 Introduction

The massively parallel computation facilitated by GPU hardware has led to a surge of successful neural network architectures for sequence labeling (Chiu and Nichols, 2016; Lample et al., 2016; Ling et al., 2015c; Ma and Hovy, 2016). While these models are expressive and accurate, they fail to fully exploit the parallelism opportunities of a GPU, and thus their speed is limited. Specifically, they employ either recurrent neural networks (RNNs) for feature extraction, or Viterbi inference in a structured output model, both of which require sequential computation across the length of the input.

Instead, parallelized runtime independent of the length of the sequence saves time and energy costs, maximizing GPU resource usage and minimizing the amount of time it takes to train and evaluate models. Convolutional neural networks (CNNs) provide exactly this property (Kalchbrenner et al., 2014; Kim, 2014). Rather than composing representations incrementally over each token in a sequence, they apply filters in parallel across the entire sequence at once. Their computational cost grows with the number of layers, but not the input size, up to the memory and threading limitations of the hardware. This provides, for example, audio generation models that can be trained in parallel (van den Oord et al., 2016).

Despite the clear computational advantages of CNNs, RNNs have become the standard method for composing deep representations of text. This is because a token encoded by a bidirectional RNN will incorporate evidence from the entire input sequence, but the CNN's representation is limited by the *receptive field* of the architecture. Specifically, in a network composed of a series of stacked convolutional layers of convolution width w, the number r of context tokens incorporated into a token's representation at a given layer l, is given by $r = l(w-1)+1$. The number of layers required to incorporate the entire input context grows linearly with the length of the sequence. To avoid this scaling, one could pool representations across the sequence, but this is not appropriate for sequence labeling, since it reduces the output resolution of the representation.

In response, this chapter presents an application of *dilated convolutions* (Yu and Koltun, 2016) for sequence labeling (Figure 3.1). For dilated convolutions, the recep-

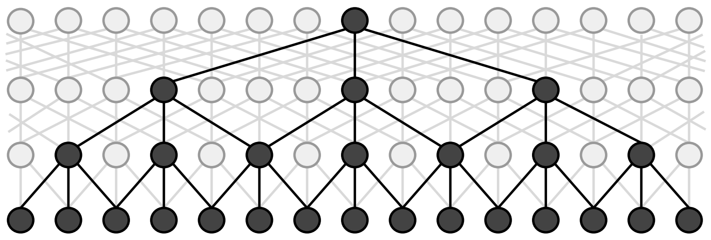

Figure 3.1: A dilated CNN block with maximum dilation width 4 and filter width 3. Neurons contributing to a single highlighted neuron in the last layer are also highlighted.

tive field can grow exponentially with the depth, with no loss in resolution at each layer and with a modest number of parameters to estimate. Like typical CNN layers, dilated convolutions operate on a sliding window of context over the sequence, but unlike conventional convolutions, the context need not be consecutive; the dilated window skips over every dilation width d inputs. By stacking layers of dilated convolutions of exponentially increasing dilation width, we can expand the size of the receptive field to cover the entire length of most sequences using only a few layers: The size of the receptive field for a token at layer l is now given by $2^{l+1} - 1$. More concretely, just four stacked dilated convolutions of width 3 produces token representations with a receptive field of 31 tokens – longer than the average sentence length (23) in the Penn TreeBank.

Our overall *iterated dilated CNN* architecture (ID-CNN) repeatedly applies the same block of dilated convolutions to token-wise representations. This parameter sharing prevents overfitting and also provides opportunities to inject supervision on intermediate activations of the network. Similar to models that use RNN features, the ID-CNN provides two methods for performing prediction: we can predict each token's label independently, or by running Viterbi inference in a chain structured graphical model.

In experiments on CoNLL 2003 and OntoNotes 5.0 English NER, we demonstrate significant speed gains of our ID-CNNs over various recurrent models, while maintaining similar F1 performance. When performing prediction using independent classification, the ID-CNN consistently outperforms a bidirectional LSTM (Bi-LSTM), and performs on par with inference in a CRF with features extracted by a Bi-LSTM (Bi-LSTM-CRF). As a feature extractor for a CRF, our model out-performs the Bi-LSTM-CRF. We also apply ID-CNNs to entire documents, where independent token classification is more accurate than the Bi-LSTM-CRF while decoding almost 8× faster. The clear accuracy gains resulting from incorporating broader context suggest that these models could similarly benefit many other context-sensitive NLP tasks which have until now been limited by the computational complexity of existing context-rich models.

3.2 Background

3.2.1 Conditional Probability Models for Tagging

In this chapter we consider the two factorizations of the conditional distribution described in §1.1.1 of Chapter 1.

Let $x = [x_1, \dots, x_T]$ be our input text and $y = [y_1, \dots, y_T]$ be per-token output tags. Let D be the domain size of each y_i. We predict the most likely y, given a conditional model $P(y|x)$.

First, we have:

$$P(y|x) = \prod_{t=1}^{T} P(y_t|F(x)), \tag{3.1}$$

where the tags are conditionally independent given some features for x.

We also consider a linear-chain CRF model that couples all of y together:

$$P(y|x) = \frac{1}{Z_x} \prod_{t=1}^{T} \psi_t(y_t|F(x))\psi_p(y_t, y_{t-1}), \tag{3.2}$$

where ψ_t is a local factor, ψ_p is a pairwise factor that scores consecutive tags, and Z_x is the partition function (Lafferty et al., 2001). To avoid overfitting, ψ_p does not depend on the timestep t or the input x in our experiments.

In Equation 3.1, $O(D)$ prediction is trivially parallelizable across the length of the sequence. However, feature extraction may not necessarily be parallelizable. For example, RNN-based features require iterative passes along the length of x. Prediction using Equation 3.2 requires global search using the $O(D^2T)$ Viterbi algorithm. While the factorization in Equation 3.2 can guarantee that certain output constraints will always be satisfied and may also have better sample complexity than Equation 3.1, it has much worse computational complexity and parallelizability than independent prediction.

In this chapter we explore parameterizations of the feature function F which allow for more complex reasoning about interactions among neighboring output tags, without requiring the expensive inference of Equation 3.2. We describe these models in more detail below.

3.3 Dilated Convolutions

Recall convolutional neural networks (CNNs) for NLP, as described in §1.1.2.3 in Chapter 1. A one-dimensional CNN layer is equivalent to applying an affine transformation, W_c to a sliding window of width r tokens on either side of each token in the sequence. Here, and throughout the chapter, we do not explicitly write the bias terms in affine transformations. The sliding-window representation c_t for each token x_t is:

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t\pm k}, \tag{3.3}$$

where $\oplus$ is vector concatenation.

Dilated convolutions perform the same operation, except rather than transforming adjacent inputs, the convolution is defined over a wider receptive field by skipping over δ inputs at a time, where δ is the dilation width. We define the dilated convolution operator:

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t \pm k\delta}. \tag{3.4}$$

A dilated convolution of width 1 is equivalent to a simple convolution. Using the same number of parameters as a simple convolution with the same radius, the $\delta > 1$ dilated convolution incorporates broader context into the representation of a token than a simple convolution.

However, it is clear that simply applying a dilated convolution to the input in order to predict the output would likely miss important evidence for labeling a token. For example, a dilated convolution of width 2 would skip over the direct neighbors of a token, which often provide important context for that token's part-of-speech tag or entity membership.

3.3.1 Multi-Scale Context Aggregation

We can leverage the ability of dilated convolutions to incorporate global context without losing important local information by stacking dilated convolutions of increasing width. First described for pixel classification in computer vision, Yu and Koltun (2016) achieve state-of-the-art results on image segmentation benchmarks by stacking dilated convolutions with exponentially increasing rates of dilation, a technique they refer to as *multi-scale context aggregation.* By feeding the outputs of each dilated convolution as the input to the next, increasingly non-local information is incorporated into each pixel's representation. Performing a dilation-1 convolution in the first layer ensures that no pixels within the receptive field of any pixel are excluded. By doubling the dilation width at each layer, the size of the receptive field grows exponentially while the number of parameters grows only linearly with the number of layers, so a pixel representation quickly incorporates rich global evidence from the entire image.

3.4 Iterated Dilated CNNs

Stacked dilated CNNs can easily incorporate global information from a whole sentence or document. For example, with a radius of 1 and 4 layers of dilated convolutions, the receptive field of each token is width 31, which exceeds the average sentence length (23) in the Penn TreeBank corpus. With a radius of size 2 and 8 layers of dilated convolutions, the receptive field exceeds 1,000 tokens, long enough to encode a full newswire document.

Unfortunately, simply increasing the depth of stacked dilated CNNs causes considerable over-fitting in our experiments. In response, we present Iterated Dilated CNNs (ID-CNNs), which instead apply the same small stack of dilated convolutions multiple times, each iterate taking as input the result of the last application. Repeatedly employing the same parameters in a recurrent fashion provides both broad receptive fields and desirable generalization capabilities. We also obtain significant accuracy gains with a training objective that strives for accurate labeling after each iterate, allowing follow-on iterates to observe and resolve dependency violations.

3.4.1 Model Architecture

The network takes as input a sequence of T vectors $\mathbf{x_t}$, and outputs a sequence of per-class scores $\mathbf{h_t}$, which serve either as the local conditional distributions of Eqn. (3.1) or the local factors ψ_t of Eqn. (3.2).

We denote the jth dilated convolutional layer of dilation width δ as $D_\delta^{(j)}$. The first layer in the network is a dilation-1 convolution $D_1^{(0)}$ that transforms the input to a representation $\mathbf{i_t}$:

$$\mathbf{i_t} = D_1^{(0)}\mathbf{x_t} \tag{3.5}$$

Next, L_c layers of dilated convolutions of exponentially increasing dilation width are applied to $\mathbf{i_t}$, folding in increasingly broader context into the embedded representation of $\mathbf{x_t}$ at each layer. Let $r()$ denote the ReLU activation function (Glorot et al., 2011). Beginning with $\mathbf{c_t}^{(0)} = \mathbf{i_t}$ we define the stack of layers with the following recurrence:

$$\mathbf{c_t}^{(j)} = r\left(D_{2^{L_c-1}}^{(j-1)}\mathbf{c_t}^{(j-1)}\right) \tag{3.6}$$

and add a final dilation-1 layer to the stack:

$$\mathbf{c_t}^{(L_c+1)} = r\left(D_1^{(L_c)}\mathbf{c_t}^{(L_c)}\right) \tag{3.7}$$

We refer to this stack of dilated convolutions as a *block* $B(\cdot)$, which has output resolution equal to its input resolution. To incorporate even broader context without over-fitting, we avoid making B deeper, and instead iteratively apply B L_b times, introducing no extra parameters. Starting with $\mathbf{b_t}^{(1)} = B(\mathbf{i_t})$:

$$\mathbf{b_t}^{(k)} = B\left(\mathbf{b_t}^{(k-1)}\right) \tag{3.8}$$

We apply a simple affine transformation W_o to this final representation to obtain per-class scores for each token $\mathbf{x_t}$:

$$\mathbf{h_t}^{(L_b)} = W_o\mathbf{b_t}^{(L_b)} \tag{3.9}$$

3.4.2 Training

Our main focus is to apply the ID-CNN as feature extraction for the first conditional model described in Sec. 3.2.1, where tags are conditionally independent given deep features, since this will enable prediction that is parallelizable across the length of the input sequence. Here, maximum likelihood training is straightforward because the likelihood decouples into the sum of the likelihoods of independent logistic regression problems for every tag, with natural parameters given by Eqn. (3.9):

$$\frac{1}{T}\sum_{t=1}^{T}\log P(y_t \mid \mathbf{h_t}^{(L_b)}) \tag{3.10}$$

We can also use the ID-CNN as input features for the CRF model (Eqn. (3.2)), where the partition function and its gradient are computed using the forward-backward algorithm.

We next present an alternative training method that helps bridge the gap between these two techniques. Sec. 3.2.1 identifies that the CRF has preferable sample complexity and accuracy since prediction directly reasons in the space of structured outputs. In response, we compile some of this reasoning in output space into ID-CNN feature extraction. Instead of explicit reasoning over output labels during inference, we train the network such that each block is predictive of output labels. Subsequent blocks learn to correct dependency violations of their predecessors, refining the final sequence prediction.

To do so, we first define predictions of the model after each of the L_b applications of the block. Let $\mathbf{h_t}^{(k)}$ be the result of applying the matrix W_o from (3.9) to $\mathbf{b_t}^{(k)}$, the output of block k. We minimize the average of the losses for each application of the block:

$$\frac{1}{L_b}\sum_{k=1}^{L_b}\frac{1}{T}\sum_{t=1}^{T}\log P(y_t \mid \mathbf{h_t}^{(k)}). \tag{3.11}$$

By rewarding accurate predictions after each application of the block, we learn a model where later blocks are used to refine initial predictions. The loss also helps reduce the vanishing gradient problem (Hochreiter, 1998) for deep architectures. Such an approach has been applied in a variety of contexts for training very deep networks in computer vision (Gülçehre and Bengio, 2016; Lee et al., 2015; Romero et al., 2014; Szegedy et al., 2015), but not to our knowledge in NLP.

We apply dropout (Srivastava et al., 2014) to the raw inputs $\mathbf{x_t}$ and to each block's output $\mathbf{b_t}^{(b)}$ to help prevent overfitting. The version of dropout typically used in practice has the undesirable property that the randomized predictor used at train time differs from the fixed one used at test time. Ma et al. (2017) present *dropout with expectation-linear regularization*, which explicitly regularizes these two predictors to behave similarly. All of our best reported results include such regularization. This is the first investigation of the technique's effectiveness for NLP, including for RNNs. We encourage its further application.

3.5 Related work

The state-of-the art models for sequence labeling include an inference step that searches the space of possible output sequences of a chain-structured graphical model, or approximates this search with a beam (Chiu and Nichols, 2016; Collobert et al., 2011; Lample et al., 2016; Ma and Hovy, 2016; Weiss et al., 2015). These outperform similar systems that use the same features, but independent local predictions. On the other hand, the greedy *sequential prediction* (Daumé III et al., 2009) approach of Ratinov and Roth (2009), which employs lexicalized features, gazetteers, and word clusters, outperforms CRFs with similar features.

LSTMs (Hochreiter and Schmidhuber, 1997) were used for NER as early as the CoNLL shared task in 2003 (Hammerton, 2003; Sang and Meulder, 2003). More recently, a wide variety of neural network architectures for NER have been proposed. Collobert et al. (2011) employ a one-layer CNN with pre-trained word embeddings, capitalization and lexicon features, and CRF-based prediction. Huang et al. (2015) achieved state-of-the-art accuracy on part-of-speech, chunking and NER using a Bi-

LSTM-CRF. Lample et al. (2016) proposed two models which incorporated Bi-LSTM-composed character embeddings alongside words: a Bi-LSTM-CRF, and a greedy stack LSTM which uses a simple shift-reduce grammar to compose words into labeled entities. Their Bi-LSTM-CRF obtained the state-of-the-art on four languages without word shape or lexicon features. Ma and Hovy (2016) use CNNs rather than LSTMs to compose characters in a Bi-LSTM-CRF, achieving state-of-the-art performance on part-of-speech tagging and CoNLL NER without lexicons. Chiu and Nichols (2016) evaluate a similar network but propose a novel method for encoding lexicon matches, presenting results on CoNLL and OntoNotes NER. Yang et al. (2016) use GRU-CRFs with GRU-composed character embeddings of words to train a single network on many tasks and languages.

In general, distributed representations for text can provide useful generalization capabilities for NER systems, since they can leverage unsupervised pre-training of distributed word representations (Collobert et al., 2011; Passos et al., 2014; Turian et al., 2010). Though our models would also likely benefit from additional features such as character representations and and lexicons, we focus on simpler models which use word-embeddings alone, leaving more elaborate input representations to future work.

In these NER approaches, CNNs were used for low-level mapping feature extraction that feeds into alternative architectures. Overall, end-to-end CNNs have mainly been used in NLP for sentence classification, where the output representation is lower resolution is lower than that of the input Kalchbrenner et al. (2014); Kim (2014); Toutanova et al. (2015); Zhang et al. (2015a). Lei et al. (2015) present a CNN variant where convolutions adaptively skip neighboring words. While the flexibility of this model is powerful, its adaptive behavior is not well-suited to GPU acceleration.

Our work draws on the use of dilated convolutions for image segmentation in the computer vision community (Chen et al., 2015; Yu and Koltun, 2016). Similar to our block, Yu and Koltun (2016) employ a *context-module* of stacked dilated convolutions of exponentially increasing dilation width. Dilated convolutions were recently applied to the task of speech generation (van den Oord et al., 2016), and concurrent with this work, Kalchbrenner et al. (2016) posted a pre-print describing a network for machine translation that uses dilated convolutions in the encoder and decoder components. We are the first to use dilated convolutions for sequence labeling.

The broad receptive field of the ID-CNN helps aggregate document-level context. Ratinov and Roth (2009) incorporate document context in their greedy model by adding features based on tagged entities within a large, fixed window of tokens. Prior work has also posed a structured model that couples predictions across the whole document (Bunescu and Mooney, 2004; Finkel et al., 2005; Sutton and McCallum, 2004).

3.6 Experimental Results

We describe experiments on two benchmark English named entity recognition datasets. On CoNLL-2003 English NER, our ID-CNN out-performs a Bi-LSTM as a feature extractor for a CRF, and with greedy decoding performs on-par with the Bi-LSTM-CRF while running at more than 14 times the speed. We also observe a performance boost in almost all models when broadening the context to incorporate entire documents, achieving an average F1 of 90.65 on CoNLL-2003, out-performing the Bi-LSTM-CRF while decoding at nearly 8 times the speed.

3.6.1 Data and Evaluation

We evaluate using labeled data from the CoNLL-2003 shared task (Sang and Meulder, 2003) and OntoNotes 5.0 (Hovy et al., 2006; Pradhan et al., 2006). Following previous work, we use the same OntoNotes data split used for co-reference resolution in the CoNLL-2012 shared task (Pradhan et al., 2012). For both datasets, we convert the IOB boundary encoding to BILOU as previous work found this encoding to result in improved performance (Ratinov and Roth, 2009). The combined entity types and boundary encodings result in 17 possible output labels in the CoNLL-2003 corpus and 74 labels in the OntoNotes corpus. As in previous work we evaluate the performance of our models using segment-level micro-averaged F1 score. Hyperparameters that resulted in the best performance on the validation set were selected via grid search.

3.6.1.1 Optimization and data pre-processing

Our models are trained end-to-end using backpropagation and mini-batched Adam (Kingma and Ba, 2015) SGD. We use dropout regularization (Srivastava et al., 2014) on the input embeddings and final dilation layer of each block, along with the dropout regularizer described in Ma et al. (2017) using a single Monte Carlo sample for each training example. We also found word dropout (Dai and Le, 2015; Lample et al., 2016) crucial for learning a high-quality representation for out-of-vocabulary words. We used the modified version of identity initialization (Le et al., 2015) reported by Yu and Koltun (2016) to initialize our dilated layers, which we found to perform the best in initial experiments compared to orthogonal and Xavier initialization (Glorot and Bengio, 2010). Since our models use the same number of filters in each dilated layer, this initialization simplifies to setting the parameters corresponding to the central token to the identity matrix, and all other parameters (corresponding to left and right context) to zero. All other layers (embeddings, projections) were initialized using normally distributed Xavier initialization.

As in previous work, we found that initializing the word embedding lookup table with pre-trained embeddings was vital to achieve good performance. In initial experiments, we found the 100-dimensional skip-n-gram (Ling et al., 2013) embeddings of Lample et al. (2016) to outperform the 50-dimensional word embeddings of Collobert et al. (2011), and so we use these 100-dimensional embeddings in all experiments. We concatenate a 5-dimensional word shape vector based on whether the token was all capitalized, not capitalized, first-letter capitalized or contained a capital letter. We preprocessed the data by replacing all digits with 0, but did not lowercase thus our embeddings are case-sensitive.

We use the parameters of the trained sentence models to initialize the parameters of the document models in order to significantly speed up the rate of convergence of the document models.

3.6.1.2 Evaluation

To select hyperparameters, we iteratively perform grid search over increasingly fine-grained settings of dropout, learning rate, Adam β_2 and ϵ parameters, gradient clipping threshold, number of dilated layers, number of repeated blocks, regularizer penalty and batch size. Since we found the variance in score between runs to vary significantly, in the last iteration of grid search, we ran each setting of parameters three times and averaged their scores on the validation set. Of these, we ran the top ten settings ten times, and took the parameters which averaged the highest F1 on the development set, and report scores on the test set using these parameters. Note that we do not in the final stage include the development set as training data as has been done in some previous work, and so do not directly compare to results from other works which do so.

We evaluate test-time speed using our top-performing trained models. All timing experiments were run on a nVidia Titan X GPU with a 2.4GHz Intel Xeon CPU. We do not include data loading, preprocessing or feature hashing in our timing since this is exactly the same across all models. Reported is the time it takes for each model to produce a sequence of labels given a sequence of integers representing the words and their capitalization. After a burn-in run to account for caching and GPU data I/O, we run each model 20 times over the development set and average these times. We do this for batch sizes ranging from 1 to 10,000.

3.6.2 Baselines

We compare our **ID-CNN** against strong LSTM and CNN baselines: a **Bi-LSTM** with local decoding, and one with CRF decoding (**Bi-LSTM-CRF**). We also compare against a non-dilated CNN architecture with the same number of convolutional layers as our dilated network (**4-layer CNN**) and one with enough layers to incorporate a receptive field of the same size as that of the dilated network (**5-layer CNN**) to demonstrate that the dilated convolutions more effectively aggregate contextual information than simple convolutions (i.e. using fewer parameters). We also compare our document-level ID-CNNs to a baseline which does not share parameters between blocks (**noshare**) and one that computes loss only at the last block, rather than after every iterated block of dilated convolutions (**1-loss**).

We do not compare with deeper or more elaborate CNN architectures for a number of reasons: 1) Fast train and test performance are highly desirable for NLP practitioners, and deeper models require more computation time 2) more complicated models tend to over-fit on this relatively small dataset and 3) most accurate deep CNN architectures repeatedly up-sample and down-sample the inputs. We do not compare to stacked LSTMs for similar reasons — a single LSTM is already slower than a 4-layer CNN. Since our task is sequence labeling, we desire a model that maintains the token-level resolution of the input, making dilated convolutions an elegant solution.

3.6.3 CoNLL-2003 English NER

3.6.3.1 Sentence-level prediction

Table 3.1 lists F1 scores of models predicting with sentence-level context on CoNLL-2003. The Viterbi-decoding Bi-LSTM-CRF and ID-CNN-CRF obtain the highest average scores, with the ID-CNN-CRF outperforming the Bi-LSTM-CRF by

Model	F1
Ratinov and Roth (2009)	86.82
Collobert et al. (2011)	86.96
Lample et al. (2016)	90.33
Bi-LSTM	89.34 ± 0.28
4-layer CNN	89.97 ± 0.20
5-layer CNN	90.23 ± 0.16
ID-CNN	90.32 ± 0.26
Collobert et al. (2011)	88.67
Passos et al. (2014)	90.05
Lample et al. (2016)	90.20
Bi-LSTM-CRF (re-impl)	90.43 ± 0.12
ID-CNN-CRF	**90.54 ± 0.18**

Table 3.1: F1 score of models observing sentence-level context. No models use character embeddings or lexicons. Top models are greedy, bottom models use Viterbi inference

0.11 points of F1 on average, and the Bi-LSTM-CRF out-performing the greedy ID-CNN by 0.11 as well. Our greedy ID-CNN outperforms all other greedy models, including the 4-layer CNN which uses the same number of parameters as the ID-CNN, and the 5-layer CNN which uses more parameters but covers the same size receptive field. All CNN models out-perform the Bi-LSTM when paired with greedy decoding, suggesting that CNNs are better feature extractors than Bi-LSTMs for independent logistic regression. When paired with Viterbi decoding, our ID-CNN out-performs the Bi-LSTM as a feature extractor, showing that the ID-CNN is also a better feature extractor for Viterbi inference.

Our ID-CNN is not only a better feature extractor than the Bi-LSTM but it is also faster. Table 3.2 lists relative decoding times on the CoNLL development set, compared to the Bi-LSTM-CRF. We report decoding times for batch size 1, giving Viterbi-decoding algorithms the advantage since much of their computational

Model	Speed
Bi-LSTM-CRF	1×
Bi-LSTM	9.92×
ID-CNN-CRF	1.28×
5-layer CNN	12.38×
ID-CNN	14.10×

Table 3.2: Relative test-time speed of sentence models, using the fastest batch size for each model.[2]

overhead comes from single-thread decoding on the CPU, and with the fastest batch size for each model.[1]

The ID-CNN model is about 6 times faster than the Bi-LSTM when decoding one sentence at a time, and with larger batch sizes is nearly 50% faster. With Viterbi decoding, the gap closes somewhat but the ID-CNN-CRF still comes out ahead, about 30% faster than the Bi-LSTM-CRF. The most vast speed improvements come when comparing the greedy ID-CNN to the Bi-LSTM-CRF – our ID-CNN is more than 14 times faster than the Bi-LSTM-CRF at test time, but only 0.11 F1 points less accurate. The 5-layer CNN, which observes the same size receptive field as the ID-CNN but with more parameters, performs at about the same speed as the ID-CNN while making less accurate predictions. With a better implementation of dilated convolutions than currently included in TensorFlow, we would expect the ID-CNN to be notably faster than the 5-layer CNN.

We emphasize the importance of the dropout regularizer of Ma et al. (2017) in Table 3.3, where we observe increased F1 for every model trained with expectation-linear dropout regularization. Dropout is important for training neural network models that

[1]At scale, speed should increase with batch size, as we could compose each batch of as many sentences of the same length as would fit in GPU memory, requiring no padding and giving CNNs and ID-CNNs even more of a speed advantage.

[2]Our ID-CNN could see up to 18× speed-up with a less naive implementation than is included in TensorFlow as of this writing.

Model	w/o DR	w/ DR
Bi-LSTM	88.89 ± 0.30	**89.34** ± 0.28
4-layer CNN	89.74 ± 0.23	**89.97** ± 0.20
5-layer CNN	89.93 ± 0.32	**90.23** ± 0.16
Bi-LSTM-CRF	90.01 ± 0.23	**90.43** ± 0.12
4-layer ID-CNN	89.65 ± 0.30	**90.32** ± 0.26

Table 3.3: Comparison of models trained with and without expectation-linear dropout regularization (DR). DR improves all models.

Model	F1
4-layer ID-CNN (sent)	90.32 ± 0.26
Bi-LSTM-CRF (sent)	90.43 ± 0.12
4-layer CNN × 3	90.32 ± 0.32
5-layer CNN × 3	90.45 ± 0.21
Bi-LSTM	89.09 ± 0.19
Bi-LSTM-CRF	90.60 ± 0.19
ID-CNN	**90.65 ± 0.15**

Table 3.4: F1 score of models trained to predict document-at-a-time. Our greedy ID-CNN model performs as well as the Bi-LSTM-CRF.

generalize well, especially on relatively small NLP datasets such as CoNLL-2003. We recommend this regularizer as a simple and helpful tool for practitioners training neural networks for NLP.

3.6.3.2 Document-level prediction

In Table 3.4 we show that adding document-level context improves every model on CoNLL-2003. When incorporating document-level context, our greedy ID-CNN model out-performs the Bi-LSTM-CRF, attaining 90.65 average F1. We believe this model out-performs the Bi-LSTM-CRF due to the ID-CNN learning a feature function better suited for representing broad context, in contrast with the Bi-LSTM which, though better than a simple RNN at encoding long memories of sequences, may reach its limit when provided with sequences more than 1,000 tokens long such as entire documents.

Model	F1
ID-CNN noshare	89.81 ± 0.19
ID-CNN 1-loss	90.06 ± 0.19
ID-CNN	**90.65 ± 0.15**

Table 3.5: Comparing ID-CNNs with 1) back-propagating loss only from the final layer (**1-loss**) and 2) untied parameters across blocks (**noshare**)

Model	Speed
Bi-LSTM-CRF	1×
Bi-LSTM	4.60×
ID-CNN	7.96×

Table 3.6: Relative test-time speed of document models (fastest batch size for each model).

We also note that our combination of training objective (Eqn. 3.11) and tied parameters (Eqn. 3.8) more effectively learns to aggregate this broad context than a vanilla cross-entropy loss or deep CNN back-propagated from the final neural network layer. Table 3.5 compares models trained to incorporate entire document context using the document baselines described in Section 3.6.2.

In Table 3.6 we show that, in addition to being more accurate, our ID-CNN model is also much faster than the Bi-LSTM-CRF when incorporating context from entire documents, decoding at almost 8 times the speed. On these long sequences, it also tags at more than 4.5 times the speed of the greedy Bi-LSTM, demonstrative of the benefit of our ID-CNNs context-aggregating computation that does not depend on the length of the sequence.

3.6.4 OntoNotes 5.0 English NER

We observe similar patterns on OntoNotes as we do on CoNLL. Table 3.7 lists overall F1 scores of our models compared to those in the existing literature. The greedy Bi-LSTM out-performs the lexicalized greedy model of Ratinov and Roth

Model	F1	Speed
Ratinov and Roth (2009)[3]	83.45	
Durrett and Klein (2014)	84.04	
Chiu and Nichols (2016)	86.19 ± 0.25	
Bi-LSTM-CRF	86.99 ± 0.22	1×
Bi-LSTM-CRF-Doc	86.81 ± 0.18	1.32×
Bi-LSTM	83.76 ± 0.10	24.44×
ID-CNN-CRF (1 block)	86.84 ± 0.19	1.83×
ID-CNN-Doc (3 blocks)	85.76 ± 0.13	21.19×
ID-CNN (3 blocks)	85.27 ± 0.24	13.21×
ID-CNN (1 block)	84.28 ± 0.10	26.01×

Table 3.7: F1 score of sentence and document models on OntoNotes.

(2009), and our ID-CNN out-performs the Bi-LSTM as well as the more complex model of Durrett and Klein (2014) which leverages the parallel co-reference annotation available in the OntoNotes corpus to predict named entities jointly with entity linking and co-reference. Our greedy model is out-performed by the Bi-LSTM-CRF reported in Chiu and Nichols (2016) as well as our own re-implementation, which appears to be the new state-of-the-art on this dataset.

The gap between our greedy model and those using Viterbi decoding is wider than on CoNLL. We believe this is due to the more diverse set of entities in OntoNotes, which also tend to be much longer – the average length of a multi-token named entity segment in CoNLL is about one token shorter than in OntoNotes. These long entities benefit more from explicit structured constraints enforced in Viterbi decoding. Still, our ID-CNN outperforms all other greedy methods, achieving our goal of learning a better feature extractor for structured prediction.

Incorporating greater context significantly boosts the score of our greedy model on OntoNotes, whereas the Bi-LSTM-CRF performs more poorly. In Table 3.7, we also list the F1 of our ID-CNN model and the Bi-LSTM-CRF model trained on entire

[3]Results as reported in Durrett and Klein (2014) as this data split did not exist at the time of publication.

document context. For the first time, we see the score decrease when more context is added to the Bi-LSTM-CRF model, though the ID-CNN, whose sentence model a lower score than that of the Bi-LSTM-CRF, sees an increase. We believe the decrease in the Bi-LSTM-CRF model occurs because of the nature of the OntoNotes dataset compared to CoNLL-2003: CoNLL-2003 contains a particularly high proportion of ambiguous entities,[4] perhaps leading to more benefit from document context that helps with disambiguation. In this scenario, adding the wider context may just add noise to the high-scoring Bi-LSTM-CRF model, whereas the less accurate dilated model can still benefit from the refined predictions of the iterated dilated convolutions.

[4]According to the ACL Wiki page on CoNLL-2003: "The corpus contains a very high ratio of metonymic references (city names standing for sport teams)"

CHAPTER 4

LINGUISTICALLY-INFORMED SELF-ATTENTION FOR SEMANTIC ROLE LABELING

Current state-of-the-art semantic role labeling (SRL) uses a deep neural network with no explicit linguistic features. However, prior work has shown that gold syntax trees can dramatically improve SRL decoding, suggesting the possibility of increased accuracy from explicit modeling of syntax. In this work, we present linguistically-informed self-attention (LISA): a neural network model that combines multi-head self-attention with multi-task learning across dependency parsing, part-of-speech tagging, predicate detection and SRL. Unlike previous models which require significant pre-processing to prepare linguistic features, LISA can incorporate syntax using merely raw tokens as input, encoding the sequence only once to simultaneously perform parsing, predicate detection and role labeling for all predicates. Syntax is incorporated by training one attention head to attend to syntactic parents for each token. Moreover, if a high-quality syntactic parse is already available, it can be beneficially injected at test time without re-training our SRL model. In experiments on CoNLL-2005 SRL, LISA achieves new state-of-the-art performance for a model using predicted predicates and standard word embeddings, attaining 2.5 F1 absolute higher than the previous state-of-the-art on newswire and more than 3.5 F1 on out-of-domain data, nearly 10% reduction in error. On ConLL-2012 English SRL we also show an improvement of more than 2.5 F1. LISA also out-performs the state-of-the-art with contextually-encoded (ELMo) word representations, by nearly 1.0 F1 on news and more than 2.0 F1 on out-of-domain text.

4.1 Introduction

Semantic role labeling (SRL) extracts a high-level representation of meaning from a sentence, labeling e.g. *who* did *what* to *whom*. Explicit representations of such semantic information have been shown to improve results in challenging downstream tasks such as dialog systems (Chen et al., 2013; Tur et al., 2005), machine reading (Berant et al., 2014; Wang et al., 2015) and translation (Bazrafshan and Gildea, 2013; Liu and Gildea, 2010).

Though syntax was long considered an obvious prerequisite for SRL systems (Levin, 1993; Punyakanok et al., 2008), recently deep neural network architectures have surpassed syntactically-informed models (He et al., 2018, 2017; Marcheggiani et al., 2017; Tan et al., 2018; Zhou and Xu, 2015b), achieving state-of-the art SRL performance with no explicit modeling of syntax. An additional benefit of these end-to-end models is that they require just raw tokens and (usually) detected predicates as input, whereas richer linguistic features typically require extraction by an auxiliary pipeline of models.

Still, recent work (He et al., 2017; Marcheggiani and Titov, 2017; Roth and Lapata, 2016) indicates that neural network models could see even higher accuracy gains by leveraging syntactic information rather than ignoring it. He et al. (2017) indicate that many of the errors made by a syntax-free neural network on SRL are tied to certain syntactic confusions such as prepositional phrase attachment, and show that while constrained inference using a relatively low-accuracy predicted parse can provide small improvements in SRL accuracy, providing a gold-quality parse leads to substantial gains. Marcheggiani and Titov (2017) incorporate syntax from a high-quality parser (Kiperwasser and Goldberg, 2016) using graph convolutional neural networks (Kipf and Welling, 2017), but like He et al. (2017) they attain only small increases over a model with no syntactic parse, and even perform worse than a syntax-free model on out-of-domain data. These works suggest that though syntax has the potential to improve neural network SRL models, we have not yet designed an architecture which maximizes the benefits of auxiliary syntactic information.

In response, we propose *linguistically-informed self-attention* (LISA): a model that combines multi-task learning (Caruana, 1993) with stacked layers of multi-head self-attention (Vaswani et al., 2017); the model is trained to: (1) jointly predict parts of speech and predicates; (2) perform parsing; and (3) attend to syntactic parse parents, while (4) assigning semantic role labels. Whereas prior work typically requires separate models to provide linguistic analysis, including most syntax-free neural models which still rely on external predicate detection, our model is truly end-to-end: earlier layers are trained to predict prerequisite parts-of-speech and predicates, the latter of which are supplied to later layers for scoring. Though prior work re-encodes each sentence to predict each desired task and again with respect to each predicate to perform SRL, we more efficiently encode each sentence only once, predict its predicates, part-of-speech tags and labeled syntactic parse, then predict the semantic roles for all predicates in the sentence in parallel. The model is trained such that, as syntactic parsing models improve, providing high-quality parses at test time will improve its performance, allowing the model to leverage updated parsing models without requiring re-training.

In experiments on the CoNLL-2005 and CoNLL-2012 datasets we show that our linguistically-informed models out-perform the syntax-free state-of-the-art. On CoNLL-2005 with predicted predicates and standard word embeddings, our single model out-performs the previous state-of-the-art model on the WSJ test set by 2.5 F1 points absolute. On the challenging out-of-domain Brown test set, our model improves substantially over the previous state-of-the-art by more than 3.5 F1, a nearly 10% reduction in error. On CoNLL-2012, our model gains more than 2.5 F1 absolute over the previous state-of-the-art. Our models also show improvements when using contextually-encoded word representations (Peters et al., 2018), obtaining nearly

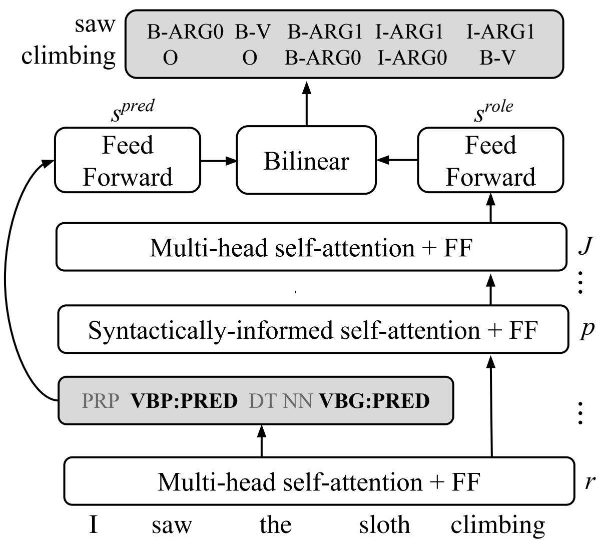
saw
climbing
B-ARG0 B-V B-ARG1 I-ARG1 I-ARG1
O O B-ARG0 I-ARG0 B-V
s^{pred}
s^{role}
Feed Forward
Bilinear
Feed Forward
Multi-head self-attention + FF
J
Syntactically-informed self-attention + FF
p
PRP VBP:PRED DT NN VBG:PRED
Multi-head self-attention + FF
r
I saw the sloth climbing

Figure 4.1: Word embeddings are input to J layers of multi-head self-attention. In layer p one attention head is trained to attend to parse parents (Figure 4.2). Layer r is input for a joint predicate/POS classifier. Representations from layer r corresponding to predicted predicates are passed to a bilinear operation scoring distinct predicate and role representations to produce per-token SRL predictions with respect to each predicted predicate.

1.0 F1 higher than the state-of-the-art on CoNLL-2005 news and more than 2.0 F1 improvement on out-of-domain text.[1]

4.2 Model

Our goal is to design an efficient neural network model which makes use of linguistic information as effectively as possible in order to perform end-to-end SRL. LISA achieves this by combining: (1) A new technique of supervising neural attention to predict syntactic dependencies with (2) multi-task learning across four related tasks.

Figure 4.1 depicts the overall architecture of our model. The basis for our model is the Transformer encoder introduced by Vaswani et al. (2017): we transform word

[1]Our implementation in TensorFlow (Abadi et al., 2015) is available at : `http://github.com/strubell/LISA`

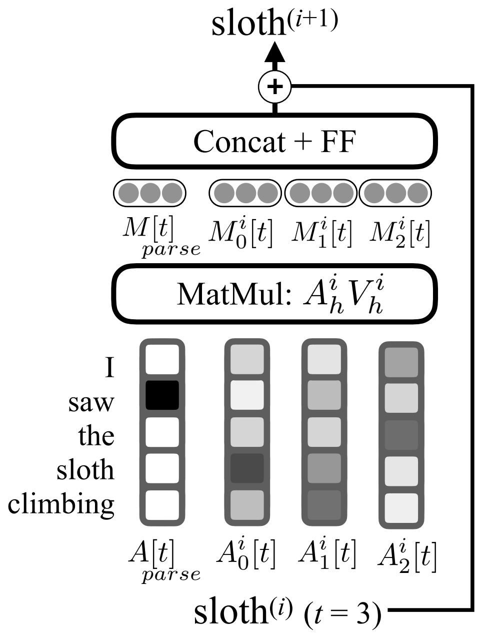

Figure 4.2: Syntactically-informed self-attention for the query word *sloth*. Attention weights A_{parse} heavily weight the token's syntactic governor, *saw*, in a weighted average over the token values V_{parse}. The other attention heads act as usual, and the attended representations from all heads are concatenated and projected through a feed-forward layer to produce the syntactically-informed representation for *sloth*.

embeddings into contextually-encoded token representations using stacked multi-head self-attention and feed-forward layers (§4.2.1).

To incorporate syntax, one self-attention head is trained to attend to each token's syntactic parent, allowing the model to use this attention head as an oracle for syntactic dependencies. We introduce this *syntactically-informed self-attention* (Figure 4.2) in more detail in §4.2.2.

Our model is designed for the more realistic setting in which gold predicates are not provided at test-time. Our model predicts predicates and integrates part-of-speech (POS) information into earlier layers by re-purposing representations closer to the input to predict predicate and POS tags using hard parameter sharing (§4.2.3). We simplify optimization and benefit from shared statistical strength derived from highly correlated POS and predicates by treating tagging and predicate detection as a single task, performing multi-class classification into the joint Cartesian product space of POS and predicate labels.

Though typical models, which re-encode the sentence for each predicate, can simplify SRL to token-wise tagging, our joint model requires a different approach to classify roles with respect to each predicate. Contextually encoded tokens are projected to distinct *predicate* and *role* embeddings (§4.2.4), and each predicted predicate is scored with the sequence's role representations using a bilinear model (Eqn. 4.3), producing per-label scores for BIO-encoded semantic role labels for each token and each semantic frame.

The model is trained end-to-end by maximum likelihood using stochastic gradient descent (§4.2.5).

4.2.1 Self-attention token encoder

The basis for our model is a multi-head self-attention token encoder, recently shown to achieve state-of-the-art performance on SRL (Tan et al., 2018), and which provides a natural mechanism for incorporating syntax, as described in §4.2.2. Our implementation replicates Vaswani et al. (2017), which is described in more detail in Chapter 1 §1.1.2.5.

The input to the network is a sequence $\mathcal{X}$ of T token representations x_t. In the standard setting these token representations are initialized to pre-trained word embeddings, but we also experiment with supplying pre-trained ELMo representations combined with task-specific learned parameters, which have been shown to substantially improve performance of other SRL models (Peters et al., 2018). For experiments with gold predicates, we concatenate a predicate indicator embedding p_t following previous work (He et al., 2017).

4.2.2 Syntactically-informed self-attention

Typically, neural attention mechanisms are left on their own to learn to attend to relevant inputs. Instead, we propose training the self-attention to attend to specific tokens corresponding to the syntactic structure of the sentence as a mechanism for passing linguistic knowledge to later layers.

Specifically, we replace one attention head with the deep bi-affine model of Dozat and Manning (2017), trained to predict syntactic dependencies. Let A_{parse} be the parse attention weights, at layer i. Its input is the matrix of token representations $S^{(i-1)}$. As with the other attention heads, we project $S^{(i-1)}$ into key, value and query representations, denoted K_{parse}, Q_{parse}, V_{parse}. Here the key and query projections correspond to *parent* and *dependent* representations of the tokens, and we allow their dimensions to differ from the rest of the attention heads to more closely follow the implementation of Dozat and Manning (2017). Unlike the other attention heads which use a dot product to score key-query pairs, we score the compatibility between K_{parse} and Q_{parse} using a bi-affine operator U_{heads} to obtain attention weights:

$$A_{parse} = \mathrm{softmax}(Q_{parse} U_{heads} K^T_{parse}) \tag{4.1}$$

These attention weights are used to compose a weighted average of the value representations V_{parse} as in the other attention heads.

We apply auxiliary supervision at this attention head to encourage it to attend to each token's parent in a syntactic dependency tree, and to encode information about the token's dependency label. Denoting the attention weight from token t to a candidate head q as $A_{parse}[t, q]$, we model the probability of token t having parent q as:

$$P(q = \mathrm{head}(t) \mid \mathcal{X}) = A_{parse}[t, q] \tag{4.2}$$

using the attention weights $A_{parse}[t]$ as the distribution over possible heads for token t. We define the root token as having a self-loop. This attention head thus emits a directed graph[2] where each token's parent is the token to which the attention A_{parse} assigns the highest weight.

We also predict dependency labels using per-class bi-affine operations between parent and dependent representations Q_{parse} and K_{parse} to produce per-label scores, with locally normalized probabilities over dependency labels y_t^{dep} given by the softmax function. We refer the reader to Dozat and Manning (2017) for more details.

This attention head now becomes an oracle for syntax, denoted $\mathcal{P}$, providing a dependency parse to downstream layers. This model not only predicts its own dependency arcs, but allows for the injection of auxiliary parse information at test time by simply setting A_{parse} to the parse parents produced by e.g. a state-of-the-art parser. In this way, our model can benefit from improved, external parsing models without re-training. Unlike typical multi-task models, ours maintains the ability to leverage external syntactic information.

4.2.3 Multi-task learning

We also share the parameters of lower layers in our model to predict POS tags and predicates. Following He et al. (2017), we focus on the end-to-end setting, where predicates must be predicted on-the-fly. Since we also train our model to predict syntactic dependencies, it is beneficial to give the model knowledge of POS information. While much previous work employs a pipelined approach to both POS tagging for dependency parsing and predicate detection for SRL, we take a multi-task learning (MTL) approach (Caruana, 1993), sharing the parameters of earlier layers in our SRL model with a joint POS and predicate detection objective. Since POS is a strong pre-

[2]Usually the head emits a tree, but we do not enforce it here.

dictor of predicates[3] and the complexity of training a multi-task model increases with the number of tasks, we combine POS tagging and predicate detection into a joint label space: For each POS tag TAG which is observed co-occurring with a predicate, we add a label of the form TAG:PREDICATE.

Specifically, we feed the representation $s_t^{(r)}$ from a layer r preceding the syntactically-informed layer p to a linear classifier to produce per-class scores r_t for token t. We compute locally-normalized probabilities using the softmax function: $P(y_t^{prp} \mid \mathcal{X}) \propto \exp(r_t)$, where y_t^{prp} is a label in the joint space.

4.2.4 Predicting semantic roles

Our final goal is to predict semantic roles for each predicate in the sequence. We score each predicate against each token in the sequence using a bilinear operation, producing per-label scores for each token for each predicate, with predicates and syntax determined by oracles $\mathcal{V}$ and $\mathcal{P}$.

First, we project each token representation $s_t^{(J)}$ to a predicate-specific representation s_t^{pred} and a role-specific representation s_t^{role}. We then provide these representations to a bilinear transformation U for scoring. So, the role label scores s_{ft} for the token at index t with respect to the predicate at index f (i.e. token t and frame f) are given by:

$$s_{ft} = (s_f^{pred})^T U s_t^{role} \tag{4.3}$$

which can be computed in parallel across all semantic frames in an entire minibatch. We calculate a locally normalized distribution over role labels for token t in frame f using the softmax function: $P(y_{ft}^{role} \mid \mathcal{P}, \mathcal{V}, \mathcal{X}) \propto \exp(s_{ft})$.

[3]All predicates in CoNLL-2005 are verbs; CoNLL-2012 includes some nominal predicates.

At test time, we perform constrained decoding using the Viterbi algorithm to emit valid sequences of BIO tags, using unary scores s_{ft} and the transition probabilities given by the training data.

4.2.5 Training

We maximize the sum of the likelihoods of the individual tasks. In order to maximize our model's ability to leverage syntax, during training we clamp $\mathcal{P}$ to the gold parse ($\mathcal{P}_G$) and $\mathcal{V}$ to gold predicates $\mathcal{V}_G$ when passing parse and predicate representations to later layers, whereas syntactic head prediction and joint predicate/POS prediction are conditioned only on the input sequence $\mathcal{X}$. The overall objective is thus:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\Big[\sum_{f=1}^{F}&\log P(y_{ft}^{role} \mid \mathcal{P}_G, \mathcal{V}_G, \mathcal{X})\\ &+\log P(y_t^{prp} \mid \mathcal{X})\\ &+\lambda_1 \log P(\text{head}(t) \mid \mathcal{X})\\ &+\lambda_2 \log P(y_t^{dep} \mid \mathcal{P}_G, \mathcal{X})\Big]\end{aligned} \tag{4.4}$$

where λ_1 and λ_2 are penalties on the syntactic attention loss.

We train the model using Nadam (Dozat, 2016) SGD combined with the learning rate schedule in Vaswani et al. (2017). In addition to MTL, we regularize our model using dropout (Srivastava et al., 2014). We use gradient clipping to avoid exploding gradients (Bengio et al., 1994; Pascanu et al., 2013). Details on optimization and hyperparameters are included in §4.4.2.

4.3 Related work

Early approaches to SRL (Johansson and Nugues, 2008; Pradhan et al., 2005; Surdeanu et al., 2007; Toutanova et al., 2008) focused on developing rich sets of linguistic features as input to a linear model, often combined with complex constrained inference e.g. with an ILP (Punyakanok et al., 2008). Täckström et al. (2015) showed that constraints could be enforced more efficiently using a clever dynamic program for exact inference. Sutton and McCallum (2005) modeled syntactic parsing and SRL jointly, and Lewis et al. (2015) jointly modeled SRL and CCG parsing.

Collobert et al. (2011) were among the first to use a neural network model for SRL, a CNN over word embeddings which failed to out-perform non-neural models. FitzGerald et al. (2015) successfully employed neural networks by embedding lexicalized features and providing them as factors in the model of Täckström et al. (2015).

More recent neural models are syntax-free. Zhou and Xu (2015b), Marcheggiani et al. (2017) and He et al. (2017) all use variants of deep LSTMs with constrained decoding, while Tan et al. (2018) apply self-attention to obtain state-of-the-art SRL with gold predicates. Like this work, He et al. (2017) present end-to-end experiments, predicting predicates using an LSTM, and He et al. (2018) jointly predict SRL spans and predicates in a model based on that of Lee et al. (2017), obtaining state-of-the-art predicted predicate SRL. Concurrent to this work, Peters et al. (2018) and He et al. (2018) report significant gains on PropBank SRL by training a wide LSTM language model and using a task-specific transformation of its hidden representations (ELMo) as a deep, and computationally expensive, alternative to typical word embeddings. We find that LISA obtains further accuracy increases when provided with ELMo word representations, especially on out-of-domain data.

Some work has incorporated syntax into neural models for SRL. Roth and Lapata (2016) incorporate syntax by embedding dependency paths, and similarly Marcheggiani and Titov (2017) encode syntax using a graph CNN over a predicted syntax tree, out-performing models without syntax on CoNLL-2009. These works are limited to incorporating partial dependency paths between tokens whereas our technique incorporates the entire parse. Additionally, Marcheggiani and Titov (2017) report that their model does not out-perform syntax-free models on out-of-domain data, a setting in which our technique excels.

MTL (Caruana, 1993) is popular in NLP, and others have proposed MTL models which incorporate subsets of the tasks we do (Collobert et al., 2011; Hashimoto et al., 2017; Peng et al., 2017; Swayamdipta et al., 2017; Zhang and Weiss, 2016), and we build off work that investigates where and when to combine different tasks to achieve the best results (Alonso and Plank, 2017; Bingel and Søgaard, 2017; Søgaard and Goldberg, 2016). Our specific method of incorporating supervision into self-attention is most similar to the concurrent work of Liu and Lapata (2018), who use edge marginals produced by the matrix-tree algorithm as attention weights for document classification and natural language inference.

The question of training on gold versus predicted labels is closely related to learning to search (Chang et al., 2015; Daumé III et al., 2009; Ross et al., 2011b) and scheduled sampling (Bengio et al., 2015), with applications in NLP to sequence labeling and transition-based parsing (Ballesteros et al., 2016; Choi and Palmer, 2011b; Goldberg and Nivre, 2012). Our approach may be interpreted as an extension of teacher forcing (Williams and Zipser, 1989) to MTL. We leave exploration of more advanced scheduled sampling techniques to future work.

4.4 Experimental results

We present results on the CoNLL-2005 shared task (Carreras and Màrquez, 2005) and the CoNLL-2012 English subset of OntoNotes 5.0 (Pradhan et al., 2006), achieving state-of-the-art results for a single model with predicted predicates on both corpora. We experiment with both standard pre-trained GloVe word embeddings (Pennington et al., 2014) and pre-trained ELMo representations with fine-tuned task-specific parameters (Peters et al., 2018) in order to best compare to prior work.

Hyperparameters that resulted in the best performance on the validation set were selected via a small grid search, and models were trained for a maximum of 4 days on one TitanX GPU using early stopping on the validation set. We convert constituencies to dependencies using the Stanford head rules v3.5 (de Marneffe and Manning, 2008).

We compare our **LISA** models to four strong baselines: For experiments using predicted predicates, we compare to He et al. (2018) and the ensemble model (**PoE**) from He et al. (2017), as well as a version of our own self-attention model which does not incorporate syntactic information (**SA**). To compare to more prior work, we present additional results on CoNLL-2005 with models given gold predicates at test time. In these experiments we also compare to Tan et al. (2018), the previous state-of-the art SRL model using gold predicates and standard embeddings.

We demonstrate that our models benefit from injecting state-of-the-art predicted parses at test time (**+D&M**) by fixing the attention to parses predicted by Dozat and Manning (2017), the winner of the 2017 CoNLL shared task (Zeman et al., 2017) which we re-train using ELMo embeddings. In all cases, using these parses at test time improves performance.

We also evaluate our model using the gold syntactic parse at test time (**+Gold**), to provide an upper bound for the benefit that syntax could have for SRL using LISA. These experiments show that despite LISA's strong performance, there remains substantial room for improvement. In §4.4.5 we perform further analysis comparing SRL models using gold and predicted parses.

4.4.1 Data and pre-processing details

We initialize word embeddings with 100d pre-trained GloVe embeddings trained on 6 billion tokens of Wikipedia and Gigaword (Pennington et al., 2014). We evaluate the SRL performance of our models using the `srl-eval.pl` script provided by the

CoNLL-2005 shared task,[4] which computes segment-level precision, recall and F1 score. We also report the predicate detection scores output by this script. We evaluate parsing using the `eval.pl` CoNLL script, which excludes punctuation.

We train distinct D&M parsers for CoNLL-2005 and CoNLL-2012. Our D&M parsers are trained and validated using the same SRL data splits, except that for CoNLL-2005 section 22 is used for development (rather than 24), as this section is typically used for validation in PTB parsing. We use Stanford dependencies v3.5 (de Marneffe and Manning, 2008) and POS tags from the Stanford CoreNLP `left3words` model (Toutanova et al., 2003). We use the pre-trained ELMo models[5] and learn task-specific combinations of the ELMo representations which are provided as input instead of GloVe embeddings to the D&M parser with otherwise default settings.

4.4.1.1 CoNLL-2012

We follow the CoNLL-2012 split used by He et al. (2018) to evaluate our models, which uses the annotations from here[6] but the subset of those documents from the CoNLL-2012 co-reference split described here[7] (Pradhan et al., 2006). This dataset is drawn from seven domains: newswire, web, broadcast news and conversation, magazines, telephone conversations, and text from the bible. The text is annotated with gold part-of-speech, syntactic constituencies, named entities, word sense, speaker, co-reference and semantic role labels based on the PropBank guidelines (Palmer et al., 2005). Propositions may be verbal or nominal, and there are 41 distinct semantic role labels, excluding continuation roles and including the predicate. We convert the se-

[4]`http://www.lsi.upc.es/~srlconll/srl-eval.pl`

[5]`https://github.com/allenai/bilm-tf`

[6]`http://cemantix.org/data/ontonotes.html`

[7]`http://conll.cemantix.org/2012/data.html`

mantic proposition and role segmentations to BIO boundary-encoded tags, resulting in 129 distinct BIO-encoded tags (including continuation roles).

4.4.1.2 CoNLL-2005

The CoNLL-2005 data (Carreras and Màrquez, 2005) is based on the original PropBank corpus (Palmer et al., 2005), which labels the Wall Street Journal portion of the Penn TreeBank corpus (PTB) (Marcus et al., 1993a) with predicate-argument structures, plus a challenging out-of-domain test set derived from the Brown corpus (Francis and Kučera, 1964). This dataset contains only verbal predicates, though some are multi-word verbs, and 28 distinct role label types. We obtain 105 SRL labels including continuations after encoding predicate argument segment boundaries with BIO tags.

4.4.2 Optimization and hyperparameters

We train the model using the Nadam (Dozat, 2016) algorithm for adaptive stochastic gradient descent (SGD), which combines Adam (Kingma and Ba, 2015) SGD with Nesterov momentum (Nesterov, 1983). We additionally vary the learning rate lr as a function of an initial learning rate lr_0 and the current training step $step$ as described in Vaswani et al. (2017) using the following function:

$$lr = lr_0 \cdot \min(step^{-0.5}, step \cdot warm^{-1.5}) \tag{4.5}$$

which increases the learning rate linearly for the first $warm$ training steps, then decays it proportionally to the inverse square root of the step number. We found this learning rate schedule essential for training the self-attention model. We only update optimization moving-average accumulators for parameters which receive gradient updates at a given step.[8]

[8]Also known as *lazy* or *sparse* optimizer updates.

	WSJ Test			Brown Test		
GloVe	P	R	F1	P	R	F1
He et al. (2017) PoE	82.0	83.4	82.7	69.7	70.5	70.1
He et al. (2018)	81.2	83.9	82.5	69.7	71.9	70.8
SA	84.17	83.28	83.72	72.98	70.1	71.51
LISA	84.07	83.16	83.61	73.32	70.56	71.91
+D&M	**85.53**	**84.45**	**84.99**	**75.8**	**73.54**	**74.66**
ELMo						
He et al. (2018)	84.8	**87.2**	86.0	73.9	**78.4**	76.1
SA	86.21	85.98	86.09	77.1	75.61	76.35
LISA	86.69	86.42	86.55	78.95	77.17	78.05
+D&M	**87.13**	86.67	**86.90**	**79.02**	77.49	**78.25**

Table 4.1: Precision, recall and F1 on the CoNLL-2005 test sets.

In all of our experiments we used initial learning rate 0.04, $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 1 \times 10^{-12}$ and dropout rates of 0.1 everywhere. We use 10 or 12 self-attention layers made up of 8 attention heads each with embedding dimension 25, with 800d feed-forward projections. In the syntactically-informed attention head, Q_{parse} has dimension 500 and K_{parse} has dimension 100. The size of *predicate* and *role* representations and the representation used for joint part-of-speech/predicate classification is 200. We train with $warm = 8000$ warmup steps and clip gradient norms to 1. We use batches of approximately 5000 tokens.

4.4.3 Semantic role labeling

Tables 4.2 and 4.1 list precision, recall and F1 on the CoNLL-2005 development and test sets using predicted predicates. For models using GloVe embeddings, our syntax-free SA model already achieves a new state-of-the-art by jointly predicting predicates, POS and SRL. LISA with its own parses performs comparably to SA, but when supplied with D&M parses LISA out-performs the previous state-of-the-art by 2.5 F1 points. On the out-of-domain Brown test set, LISA also performs

	Predicted predicates			Gold predicates		
GloVe	P	R	F1	P	R	F1
He et al. (2017) PoE	81.8	81.2	81.5	81.8	81.2	81.5
He et al. (2018)	81.3	81.9	81.6	—	—	—
Tan et al. (2018)	—	—	—	81.3	81.9	81.6
SA	83.52	81.28	82.39	83.12	82.81	82.97
LISA	83.1	81.39	82.24	83.6	83.74	83.67
+D&M	**84.59**	**82.59**	**83.58**	**85.04**	**85.51**	**85.27**
+Gold	*87.91*	*85.73*	*86.81*	*89.11*	*89.38*	*89.25*
ELMo						
He et al. (2018)	84.9	**85.7**	85.3			
SA	85.78	84.74	85.26			
LISA	**86.07**	84.64	**85.35**			
+D&M	85.83	84.51	85.17			
+Gold	*88.51*	*86.77*	*87.63*			

Table 4.2: Precision, recall and F1 on the CoNLL-2005 development set with predicted and gold predicates.

comparably to its syntax-free counterpart with its own parses, but with D&M parses LISA performs exceptionally well, more than 3.5 F1 points higher than He et al. (2018). Incorporating ELMo embeddings improves all scores. The gap in SRL F1 between models using LISA and D&M parses is smaller due to LISA's improved parsing accuracy (see §4.4.4), but LISA with D&M parses still achieves the highest F1: nearly 1.0 absolute F1 higher than the previous state-of-the art on WSJ, and more than 2.0 F1 higher on Brown. In both settings LISA leverages domain-agnostic syntactic information rather than over-fitting to the newswire training data which leads to high performance even on out-of-domain text.

To compare to more prior work we also evaluate our models in the artificial setting where gold predicates are provided at test time. For fair comparison we use GloVe embeddings, provide predicate indicator embeddings on the input and re-encode the sequence relative to each gold predicate. Here LISA still excels: with D&M parses,

	WSJ Test			Brown Test		
GloVe	P	R	F1	P	R	F1
He et al. (2018)	84.2	83.7	83.9	74.2	73.1	73.7
Tan et al. (2018)	84.5	85.2	84.8	73.5	74.6	74.1
SA	84.7	84.24	84.47	73.89	72.39	73.13
LISA	84.72	84.57	84.64	74.77	74.32	74.55
+D&M	**86.02**	**86.05**	**86.04**	**76.65**	**76.44**	**76.54**

Table 4.3: Precision, recall and F1 on the CoNLL-2005 test set with gold predicates.

LISA out-performs the previous state-of-the-art by more than 2 F1 on both WSJ and Brown.

Table 4.4 reports precision, recall and F1 on the CoNLL-2012 test set. We observe performance similar to that observed on ConLL-2005: Using GloVe embeddings our SA baseline already out-performs He et al. (2018) by nearly 1.5 F1. With its own parses, LISA slightly under-performs our syntax-free model, but when provided with stronger D&M parses LISA out-performs the state-of-the-art by more than 2.5 F1. Like CoNLL-2005, ELMo representations improve all models and close the F1 gap between models supplied with LISA and D&M parses. On this dataset ELMo also substantially narrows the difference between models with- and without syntactic information. This suggests that for this challenging dataset, ELMo already encodes much of the information available in the D&M parses. Yet, higher accuracy parses could still yield improvements since providing gold parses increases F1 by 4 points even with ELMo embeddings.

We further analyze the performance of our models on the CoNLL-2012 dataset divided by domain[9] Table 4.5 lists F1 scores on the development and test sets for each domain as well as overall, for models trained only on the newswire portion of

[9]What we refer to as *domains* are referred to as *genres* in the OntoNotes/CoNLL-2012 corpus.

	Dev			Test		
GloVe	P	R	F1	P	R	F1
He et al. (2018)	79.2	79.7	79.4	79.4	80.1	79.8
SA	82.32	79.76	81.02	82.55	80.02	81.26
LISA	81.77	79.65	80.70	81.86	79.56	80.70
+D&M	**82.97**	**81.14**	**82.05**	**83.3**	**81.38**	**82.33**
+Gold	*87.57*	*85.32*	*86.43*	—	—	—
ELMo	P	R	F1	P	R	F1
He et al. (2018)	82.1	**84.0**	83.0	81.9	**84.0**	82.9
SA	84.35	82.14	83.23	**84.39**	82.21	83.28
LISA	**84.19**	82.56	**83.37**	83.97	82.29	83.12
+D&M	84.09	82.65	83.36	84.14	82.64	**83.38**
+Gold	*88.22*	*86.53*	*87.36*	—	—	—

Table 4.4: Precision, recall and F1 on the CoNLL-2012 development and test sets. Italics indicate a synthetic upper bound obtained by providing a gold parse at test time.

the corpus (nw; upper portion of tables), and those trained on all domains, all using GloVe embeddings. We also experiment with adding external parses from the D&M parser trained on the full data (D&M$_{all}$), as well as one trained only on the newswire portion of the dataset (D&M$_{nw}$), with UAS also reported for all domains and models. As expected we find that performance degrades substantially (about 10 F1 absolute overall) across all models when limiting training to just newswire text. F1 scores of all models trained on newswire only across all domains are depicted in a bar chart in Figure 4.3. We observe the same trend as we observed overall, where adding D&M parses improves SRL F1. Adding a D&M parser trained on all domains further improves SRL F1 over the D&M parser trained on only news (except for within the news domain), and adding gold parses further improves over that. Across domains, most see similar improvement of more than 2 F1 absolute (up to 2.4) by adding D&M parses, with the exception of telephone conversations data (tc), which improves by just under 1 F1, and biblical text (pt), which improves by 1.7. We hypothesize that the telephone conversations data, transcriptions of highly informal spoken conversations,

		Dev F1							
	GloVe	bc	bn	mz	nw	pt	tc	wb	Overall
	Support	9770	7184	4567	15,078	6775	4845	5687	53,906
UAS	LISA	86.49	90.24	90.93	92.15	94.01	82.91	88.54	89.91
	D&M$_{nw}$	90.84	93.83	94.92	95.22	96.92	85.74	93.22	93.50
	D&M$_{all}$	94.04	95.45	95.57	95.71	98.33	92.78	94.65	95.29
nw	SA	66.80	70.98	66.03	75.45	79.47	70.21	67.82	71.74
	LISA	66.59	71.03	65.71	75.37	79.46	69.93	67.25	71.57
	+D&M$_{nw}$	69.11	73.38	68.22	77.65	81.14	71.13	70.19	73.81
	+D&M$_{all}$	69.86	74.22	69.18	77.65	81.97	71.50	71.15	74.38
	+Gold	*71.22*	*76.17*	*70.74*	*80.12*	*82.72*	*72.38*	*73.68*	*76.15*
UAS	LISA	91.86	93.57	93.85	93.37	97.94	90.62	91.81	93.23
all	SA	80.21	80.51	77.46	77.96	93.09	82.99	77.67	81.02
	LISA	79.63	79.65	77.08	77.82	92.92	83.22	77.65	80.72
	+D&M$_{all}$	80.25	81.15	79.25	80.02	93.20	84.49	79.80	82.05
	+Gold	*85.78*	*85.78*	*82.78*	*84.18*	*94.95*	*88.79*	*85.64*	*86.51*

		Test F1							
		bc	bn	mz	nw	pt	tc	wb	Overall
	Support	11,875	7520	5672	13,524	7220	4097	5633	55,541
UAS	LISA	88.02	89.89	90.43	92.07	94.26	83.99	88.69	90.13
	D&M$_{nw}$	92.24	93.47	94.76	94.95	97.01	87.40	92.69	93.67
	D&M$_{all}$	94.80	95.38	95.89	95.07	98.23	93.62	94.45	95.30
nw	SA	68.55	71.36	66.27	74.97	80.05	67.64	67.94	71.65
	LISA	68.77	70.77	67.38	74.97	80.34	67.59	68.11	71.77
	+D&M$_{nw}$	70.79	73.36	69.70	77.50	82.12	69.41	70.63	74.03
	+D&M$_{all}$	71.39	74.17	70.64	77.45	82.62	70.67	71.53	74.60
UAS	LISA	92.63	93.53	93.56	93.20	97.65	92.14	91.76	93.38
all	SA	81.59	79.21	78.11	78.24	92.38	82.81	78.14	81.26
	LISA	80.91	78.81	77.19	77.27	92.24	81.92	77.71	80.60
	+D&M$_{all}$	81.78	80.88	79.98	79.58	92.90	83.89	80.36	82.33

Table 4.5: F1 of GloVe models trained on only the news portion (nw) or all domains, tested on all domains in the CoNLL-2012 development and test data. Italics indicate a synthetic upper bound obtained by providing a gold parse at test time.

		Dev F1							
ELMo		bc	bn	mz	nw	pt	tc	wb	Overall
Support		9770	7184	4567	15,078	6775	4845	5687	53,906
UAS	LISA	90.22	93.54	94.23	94.71	96.27	85.05	92.43	92.93
	D&M$_{nw}$	90.84	93.83	94.92	95.22	96.92	85.74	93.22	93.50
	D&M$_{all}$	94.04	95.45	95.57	95.71	98.33	92.78	94.65	95.29
nw	SA	72.02	76.43	71.76	80.01	84.08	72.01	73.69	76.52
	LISA	72.20	76.27	72.79	80.44	83.36	72.81	74.07	76.75
	+D&M$_{nw}$	72.66	76.72	72.90	80.87	83.73	72.63	74.81	77.13
	+D&M$_{all}$	73.32	77.73	73.78	80.94	84.39	73.59	75.57	77.73
	+Gold	*74.58*	*79.60*	*75.48*	*83.47*	*85.21*	*74.49*	*78.23*	*79.52*
UAS	LISA	93.29	95.47	95.54	95.15	98.35	92.25	94.12	94.89
all	SA	80.26	82.93	81.57	81.26	93.50	85.06	81.36	83.24
	LISA	80.93	83.39	81.23	81.39	93.30	85.49	82.21	83.53
	+D&M$_{all}$	82.20	83.00	81.27	81.32	93.39	85.72	82.19	83.30
	+Gold	*84.93*	*86.30*	*84.08*	*84.71*	*94.96*	*88.80*	*86.55*	*87.36*

		Test F1							
		bc	bn	mz	nw	pt	tc	wb	Overall
Support		11,875	7520	5672	13,524	7220	4097	5633	55,541
UAS	LISA	91.30	93.12	93.67	94.32	96.27	86.36	92.05	92.93
	D&M$_{nw}$	92.24	93.47	94.76	94.95	97.01	87.40	92.69	93.67
	D&M$_{all}$	94.80	95.38	95.89	95.07	98.23	93.62	94.45	95.30
nw	SA	73.97	76.14	73.07	79.68	84.66	72.03	73.35	76.76
	LISA	74.82	76.21	72.85	80.08	83.74	71.85	73.84	76.93
	+D&M$_{nw}$	75.23	76.83	73.32	80.28	84.08	72.44	74.48	77.36
	+D&M$_{all}$	75.98	77.57	73.91	80.19	84.76	73.36	75.15	77.88
UAS	LISA	94.00	95.78	95.38	94.61	98.04	93.08	94.21	94.82
all	SA	81.73	82.62	81.02	80.88	92.70	84.79	81.66	83.23
	LISA	82.24	82.28	81.05	81.31	92.75	84.38	81.50	83.36
	+D&M$_{all}$	83.61	82.46	81.42	81.31	92.85	84.97	82.52	83.43

Table 4.6: F1 of ELMo models trained on only the news portion (nw) or all domains, tested on all domains in the CoNLL-2012 development and test data. Italics indicate a synthetic upper bound obtained by providing a gold parse at test time.

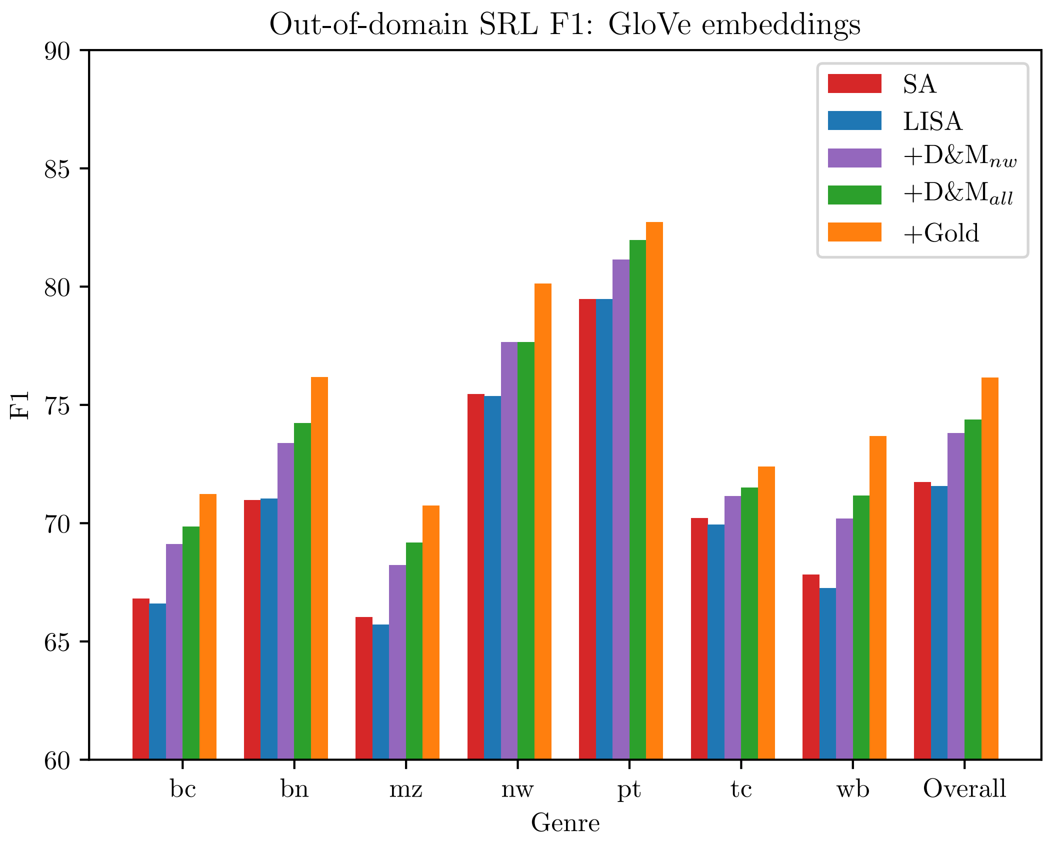

Figure 4.3: Bar chart depicting F1 scores on different CoNLL-2012 genres for a model trained only on the newswire (nw) portion of the data, with GloVe embeddings.

sees the least improvement because it is the least syntactically similar to the news data the models were trained on. The biblical text, on the other hand, boasts the highest overall scores, and we hypothesize that it sees already-high scores and reduced improvement resulting from syntax due to its relatively simple syntactic structure. SRL F1 after adding gold parse information is still well below SRL F1 for models which observed SRL and syntax training data across all domains, suggesting that though incorporating parse information does help over a model without, even with oracle syntax this model's SRL performance is still substantially limited (about 8 F1 difference) by observing only news domain training data.

Data	Model	POS	UAS	LAS
WSJ	LISA +GloVe	96.92	94.92	91.87
	LISA +ELMo	97.80	96.28	93.65
	D&M +ELMo	—	96.48	94.40
Brown	LISA +GloVe	94.26	90.31	85.82
	LISA +ELMo	95.77	93.36	88.75
	D&M +ELMo	—	92.56	88.52
CoNLL-12	LISA +GloVe	96.81	93.35	90.42
	LISA +ELMo	98.11	94.84	92.23
	D&M +ELMo	—	94.99	92.59

Table 4.7: Parsing (labeled and unlabeled attachment) and part-of-speech accuracies attained by the models used in SRL experiments on test datasets.

We also report the same suite of scores for models trained with ELMo embeddings in Table 4.6 and Figure 4.4. Here we also see a large difference (about 6-7 F1) in F1 between models trained on only news versus those trained on all domains, though it is smaller than in the GloVe models. As with the overall results, increases from adding syntax are far less dramatic in this setting. Adding syntax from a parser trained on all domains always helps, but SRL F1 improvements from a D&M parser trained on only news data are less consistent, and smaller than in the model trained with GloVe embeddings. Here, improvements hover around 0.6–1.0 F1, with the exception of the biblical text, where adding syntax trained on news actually hurts SRL F1 by 0.35 F1 absolute. This could suggest that the ELMo model already captures the relatively simple syntax of the bible better than a supervised model trained on news. That ELMo so substantially closes the gap between models with and without syntax is evidence that such large pre-trained models provide a promising framework for improving NLP performance on domains without supervised data. But, the fact that incorporating explicit syntax still helps in many cases suggests that these models may be improved by better modeling of linguistic structure.

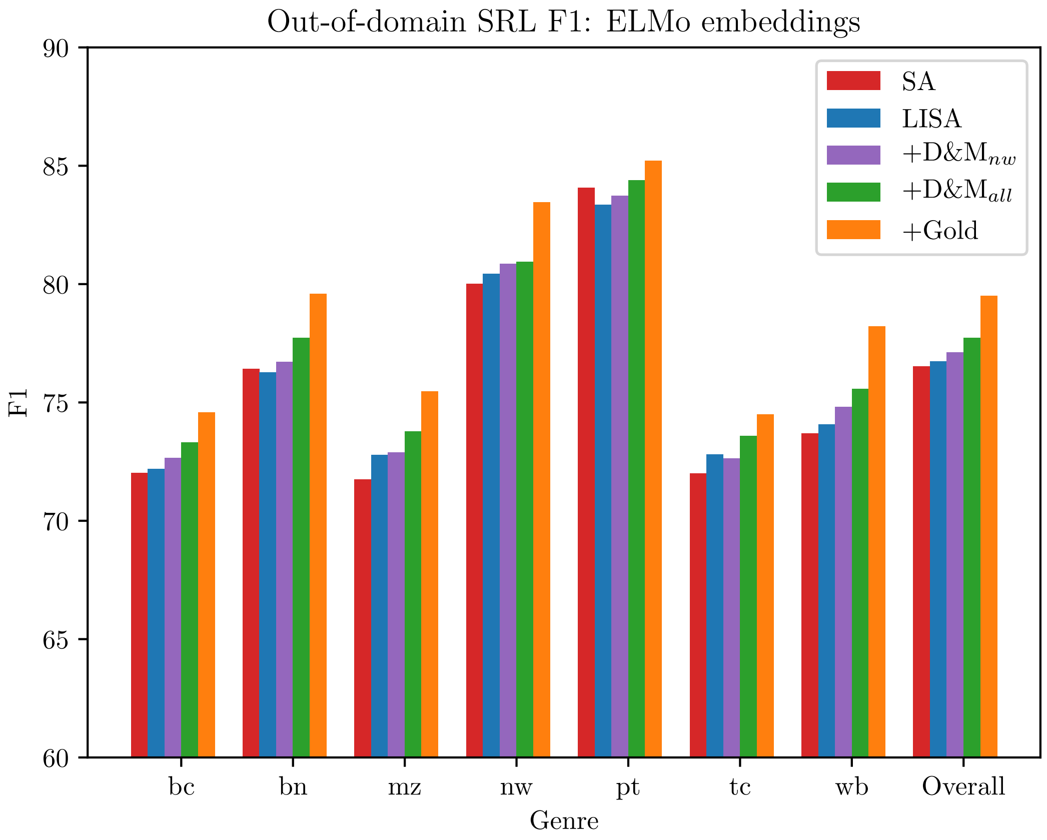

Figure 4.4: Bar chart depicting F1 scores on different CoNLL-2012 genres for a model trained only on the newswire (nw) portion of the data, with ELMo embeddings.

	Model	P	R	F1
WSJ	He et al. (2017)	94.5	98.5	96.4
	LISA	98.9	97.9	98.4
Brown	He et al. (2017)	89.3	95.7	92.4
	LISA	95.5	91.9	93.7
CoNLL-12	LISA	99.8	94.7	97.2

Table 4.8: Predicate detection precision, recall and F1 on CoNLL-2005 and CoNLL-2012 test sets.

4.4.4 Parsing, part-of-speech and predicate detection

We first report the labeled and unlabeled attachment scores (LAS, UAS) of our parsing models on the CoNLL-2005 and 2012 test sets (Table 4.7) with GloVe (+GloVe) and ELMo (+ELMo) embeddings. D&M achieves the best scores. Still, LISA's GloVe UAS is comparable to popular off-the-shelf dependency parsers such as spaCy,[10] and with ELMo embeddings comparable to the standalone D&M parser. The difference in parse accuracy between LISA+GloVe and D&M likely explains the large increase in SRL performance we see from decoding with D&M parses in that setting.

In Table 4.8 we present predicate detection precision, recall and F1 on the CoNLL-2005 and 2012 test sets. SA and LISA with and without ELMo attain comparable scores so we report only LISA+GloVe. We compare to He et al. (2017) on CoNLL-2005, the only cited work reporting comparable predicate detection F1. LISA attains high predicate detection scores, above 97 F1, on both in-domain datasets, and outperforms He et al. (2017) by 1.5-2 F1 points even on the out-of-domain Brown test set, suggesting that multi-task learning works well for SRL predicate detection.

[10]At the time of writing, spaCy reports 94.48 UAS on WSJ using Stanford dependencies v3.3: `https://spacy.io/usage/facts-figures`

CoNLL-2005	L+/D+	L–/D+	L+/D–	L–/D–
Proportion	26%	12%	4%	56%
SA	79.29	75.14	75.97	75.08
LISA	79.51	74.33	79.69	75.00
+D&M	79.03	76.96	77.73	76.52
+Gold	*79.61*	*78.38*	*81.41*	*80.47*

CoNLL-2012	L+/D+	L-/D+	L+/D-	L-/D-
Proportion	37%	10%	4%	49%
SA	76.12	75.97	82.25	65.78
LISA	76.37	72.38	85.50	65.10
+D&M	76.33	79.65	75.62	66.55
+Gold	*76.71*	*80.67*	*86.03*	*72.22*

Table 4.9: Average SRL F1 on CoNLL-2005 and CoNLL-2012 for sentences where LISA (L) and D&M (D) parses were correct (+) or incorrect (–).

4.4.5 Analysis

First we assess SRL F1 on sentences divided by parse accuracy. Table 4.9 lists average SRL F1 (across sentences) for the four conditions of LISA and D&M parses being correct or not (**L**±, **D**±). Both parsers are correct on 26% of sentences. Here there is little difference between any of the models, with LISA models tending to perform slightly better than SA. Both parsers make mistakes on the majority of sentences (57%), difficult sentences where SA also performs the worst. These examples are likely where gold and D&M parses improve the most over other models in overall F1: Though both parsers fail to correctly parse the entire sentence, the D&M parser is less wrong (87.5 vs. 85.7 average LAS), leading to higher SRL F1 by about 1.5 average F1.

We also compare the impact of Viterbi decoding with LISA, D&M, and gold syntax trees (Table 4.10), finding the same trends across both datasets. We find that Viterbi has nearly the same impact for LISA, D&M and gold parses: Gold parses provide little improvement over predicted parses in terms of BIO label consistency.

CoNLL-2005	Greedy F1	Viterbi F1	Δ F1
LISA	81.99	82.24	+0.25
+D&M	83.37	83.58	+0.21
+Gold	*86.57*	*86.81*	*+0.24*

CoNLL-2012	Greedy F1	Viterbi F1	Δ F1
LISA	80.11	80.70	+0.59
+D&M	81.55	82.05	+0.50
+Gold	*85.94*	*86.43*	*+0.49*

Table 4.10: Comparison of development F1 scores with and without Viterbi decoding at test time.

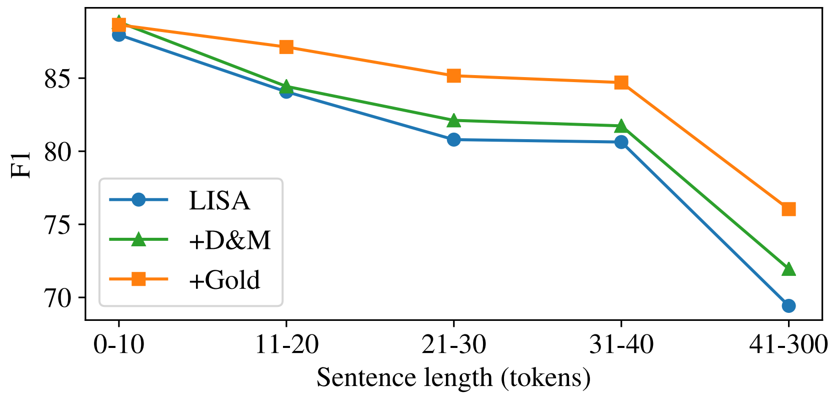

Figure 4.5: F1 score as a function of sentence length.

We also assess SRL F1 as a function of sentence length and distance from span to predicate. In Figure 4.5 we see that providing LISA with gold parses is particularly helpful for sentences longer than 10 tokens. This likely directly follows from the tendency of syntactic parsers to perform worse on longer sentences. With respect to distance between arguments and predicates, (Figure 4.6), we do not observe this same trend, with all distances performing better with better parses, especially gold.

Following He et al. (2017), we next apply a series of corrections to model predictions in order to understand which error types the gold parse resolves: e.g. *Fix Labels*

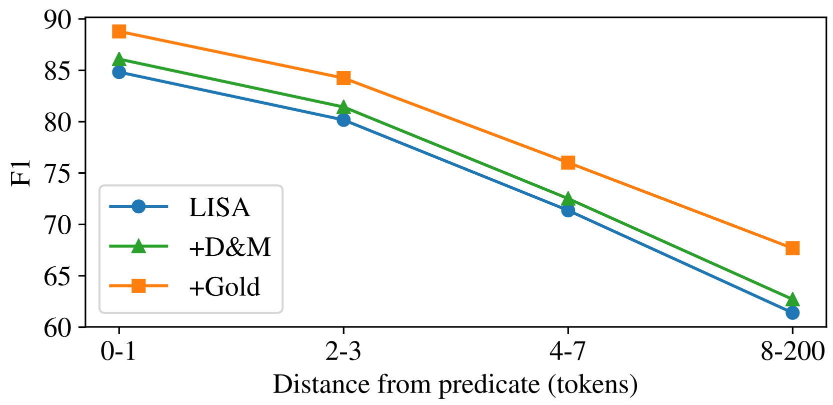

Figure 4.6: CoNLL-2005 F1 score as a function of the distance of the predicate from the argument span.

fixes labels on spans matching gold boundaries, and *Merge Spans* merges adjacent predicted spans into a gold span.[11]

In Figure 4.7 we see that much of the performance gap between the gold and predicted parses is due to span boundary errors (*Merge Spans*, *Split Spans* and *Fix Span Boundary*), which supports the hypothesis proposed by He et al. (2017) that incorporating syntax could be particularly helpful for resolving these errors. He et al. (2017) also point out that these errors are due mainly to prepositional phrase (PP) attachment mistakes. We also find this to be the case: Figure 4.8 shows a breakdown of split/merge corrections by phrase type. Though the number of corrections decreases substantially across phrase types, the proportion of corrections attributed to PPs remains the same (approx. 50%) even after providing the correct PP attachment to the model, indicating that PP span boundary mistakes are a fundamental difficulty for SRL.

[11]Refer to He et al. (2017) for a detailed explanation of the different error types.

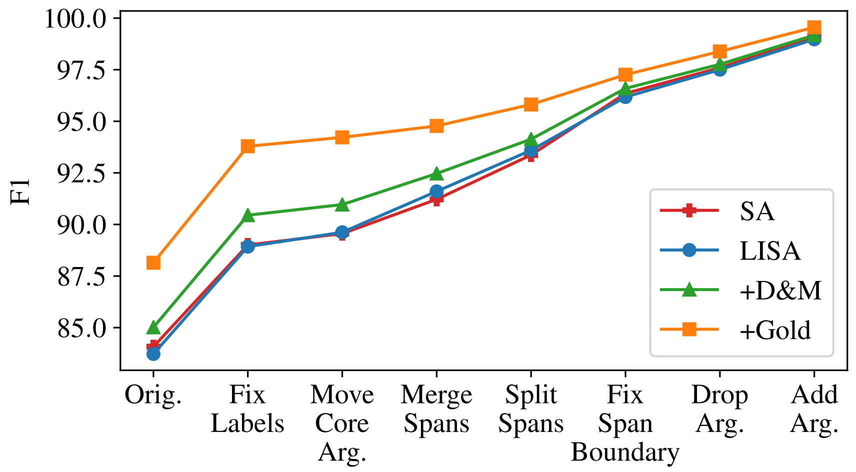

Figure 4.7: Performance of CoNLL-2005 models after performing corrections from He et al. (2017).

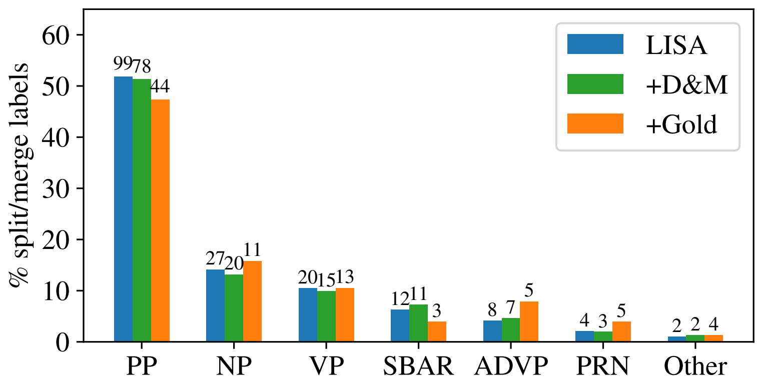

Figure 4.8: Percent and count of split/merge corrections performed in Figure 4.7, by phrase type.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

The over-arching motivation for the work in this thesis is to provide a fast NLP pipeline with robust performance across data domains. This has indeed been achieved for an English NLP pipeline consisting of part-of-speech tagging, predicate detection, named entity recognition, syntactic parsing and semantic role labeling. Though the work in this thesis covers a range of common NLP annotations, of course practitioners desire even more analysis of increasing complexity and requiring increasingly larger amounts of context, e.g. coreference resolution, machine translation, and reading comprehension. They also desire more drastic generalization, to very disparate domains and even new languages. And of course, as in the past, further increasing the efficiency of training and inference in NLP models is going to require an ongoing effort to adapt models in response to continuing improvements, and resulting architectural limitations, in specialized computer hardware for machine learning. In the following sections, I propose some promising and important directions for future research stemming from the body of work described in this thesis.

5.1 Energy efficient NLP

A cornerstone of scaling NLP, and this thesis, is building models that can be tractably run on very large corpora, and there is plenty of room for continued innovation in this direction. There has recently been great success across many NLP tasks by pre-training word representations using a deep neural network trained with a language modeling objective on vast amounts of unlabeled text, such as ELMo (Peters et al.,

2018) and BERT (Devlin et al., 2019). These models are likely to become a basic building block for all future high-accuracy NLP models due to their ability to substantially increase accuracy regardless of the end task, without requiring additional annotation. But pre-training these models requires immense time and equipment resources, e.g. weeks on arrays of specialized hardware. Incorporating these models into existing systems also substantially slows training since it still requires encoding tokens in these deep models, in addition to the task-specific modeling that is already being performed. The required resources are feasible for large tech companies but less so to researchers or smaller groups wishing to analyze text. In order to make the future of state-of-the-art NLP accessible to all, we will need to develop techniques for speeding up training and inference using these models based on intuitions similar to those that described in Chapter 2: It's simply not necessary to do the same amount of computation for every single token. Recent work has already shown that these models can be pruned to help test-time speed (Liu et al., 2018), but training time is also important — one of the appealing qualities of these models is that they can be trained on unlabeled data from any domain in order to improve text representations in that domain, and thus improve the generalization of NLP models. I'm confident we can develop methods to make these models more computationally feasible during training and decoding.

Doing so will require methodological advances in architecture and learning algorithms that are cognizant of the strengths and limitations of new tensor processing hardware. An example of this is the work described in Chapter 3, which is efficient on GPUs in particular, and the self-attention architecture that LISA (Chapter 4) is based on is designed to be particularly fast on Google's TPUs (Jouppi et al., 2017; Vaswani et al., 2017), and is enjoying great success thanks to this. Moving forward, we can expect to see an explosion of new hardware with the potential to further accelerate deep learning, bring it to smaller and cheaper devices, and vastly improve its high carbon footprint. But, simply executing the same models on new hardware is not going to achieve that potential. Having more efficient machine learning is going to require fundamental research to develop models and training techniques that are designed with the underlying hardware architecture in mind. I believe the NLP community should make a concerted effort to increase collaborations with colleagues working on systems and hardware to push this research forward.

My work so far on efficient NLP has focused on tasks which are already among the fastest in NLP, such as tagging and parsing, but going forward I believe many of the same concepts can be adapted to more complicated tasks typically requiring larger, slower models such as machine translation, and there is even more potential to increase efficiency in these models. One promising direction is developing better techniques for *non-autoregressive* machine translation (Gu et al., 2018; Libovický and Helcl, 2018). State-of-the-art machine translation systems perform autoregressive prediction of the target sequence, predicting each output token one-at-a-time, conditioned on the previous prediction. This is the most substantial speed bottleneck in machine translation. Autoregressive prediction encourages local coherence of the output, but as we've shown in previous work (Strubell et al., 2017), today's powerful deep neural networks are capable of encoding beliefs about adjacent token labels and enforcing constraints between them. I believe that with better modeling we should be able to perform much faster machine translation by reasoning over and predicting the whole output sequence in parallel. The main hurdle to this exact problem is learning to predict the length of the target sentence given the source. Once the model has built a per-token representation for the target sentence, it can reason over those representations just as it would the source. Current models predict length either via an external, non-differentiable model (Gu et al., 2018), or are left to learn these mappings on their own but relying on expensive inference techniques (Lee et al., 2018; Libovický and Helcl, 2018). Incorporating syntax has already been shown to help autoregressive neural MT systems by improving source-target alignment (Aharoni and Goldberg, 2017; Eriguchi et al., 2017). I believe that clever modeling of more explicit structure between the input and target sequence, perhaps informed by syntax, could help non-autoregressive models in particular to better learn this mapping, facilitating state-of-the-art MT accuracy with multiplicative speed-ups.

5.2 Optimization for extreme generalization

One of the biggest challenges limiting the ability of NLP tools to effectively extract actionable representations from diverse texts is limited generalization to new text domains and languages. Most NLP evaluation is limited to English-language mainstream news or Wikipedia articles, and the neural network models typically achieving state-of-the-art accuracy in NLP today have millions or billions of parameters, and are therefore data-hungry and especially prone to the over-fitting that causes lack of generalization.

Chapter 4 documented promising generalization stemming simply from providing later layers in LISA with syntactic parse information. In those experimental results, the out-of-domain evaluation comes from a subsection of the Brown corpus (Francis, 1965) consisting of text from English novels, and the non-newswire portions of the OntoNotes corpus (Pradhan et al., 2006), which include broadcast conversations, telephone calls, biblical text, and discussion forums. While this text does differ stylistically and topically from the training text, articles from the Wall Street Journal from the year 1989, most of it does not differ as drastically as e.g. very informal text, social media text, or text in another language. I hypothesize that this simpler technique that worked well for more similar domains will not work as well for these more drastically differing domains,[1] and that to achieve more drastic generalization will require

[1]Indeed, results in Chapter 4 suggest that the most distant domain from news, transcribed telephone conversations, receives the worst SRL accuracy from a model trained on news.

a training objective that explicitly rewards better generalization. One promising approach is to better adapt the idea of domain adversarial training of neural networks (Ganin et al., 2016) to problems in language. The general intuition behind domain adversarial methods is that if you can learn a mapping of inputs from distinct domains to a domain-agnostic representation, labeled data from only one of the domains is required to train a classifier which takes the domain-agnostic representation as input. More specifically, domain-adversarial training works by adding a *discriminator*, a classifier that takes as input the representation that we wish to be domain- or, in our case, language-agnostic, and attempts to discriminate between the source language, in our case English, and the target, in our case any of the four other languages. Since we desire that representations for which this classifier performs poorly, we maximize its loss rather than minimizing, which can be achieved through a *gradient reversal layer*: In the forward pass, this layer is an identity function; in the backward pass, this layer reverses the gradients and multiplies by a penalty, to control the contribution of the discriminator to the overall loss. Such a layer $D_\lambda(\cdot)$, defined formally as follows:

$$D_\lambda(x_t) = x_t; \qquad \frac{dD_\lambda}{dx} = -\lambda I \tag{5.1}$$

This formulation allows us to minimize the global loss as usual, and has been found to work well in practice (Ganin et al., 2016).

Domain adversarial training has been successfully applied to text classification tasks such as sentiment analysis, where the mapping typically boils down to learning that different domains have different sets of words which correspond to positive or negative sentiment. Previous work has found success in adversarial training to transfer English annotation to Arabic for question similarity re-ranking in community question answering (Joty et al., 2017), and for transferring part-of-speech tagging from newswire to informal social media text (Gui et al., 2017), more positive evidence for the success of this proposed work. I believe we can develop similar techniques to better syntactically or semantically parse e.g. biomedical research article text by transferring annotations from data-rich domains such as news, or from languages such as English to the more data-scarce languages spoken by most of the world. To do so, we will have to develop a technique for learning domain-agnostic representations of tokens and phrases in context. Approaches could incorporate linguistic theory into adversarial modeling, or align corpus-level co-occurrence patterns between symbols in different languages or domains.

5.3 Optimization and inference for multi-task learning

As discussed in §5.5 I believe that building *joint* NLP models that consider many tasks at once is a promising way of building models that provide a fast, accurate and robust NLP pipeline. As we showed with LISA in Chapter 4, modeling related tasks together in the right way leads to improved accuracy and generalization over single-task models. Additionally, by eliminating redundant computation across tasks, joint models are also faster than their single-model counterparts when multiple task outputs are desired. One challenge to training multi-task models is how to best trade off the many training objectives, without doing expensive structured inference: the LISA model is tuned to maximize SRL performance on a held-out set, and as a result obtained state-of-the-art SRL performance, but its accuracy for syntactic parsing is not state-of-the-art. I hypothesize that this is the result of poor optimization in the face of many competing objectives. Current multi-task learning approaches either (1) sum the losses with a fixed penalty on each, tuned as a hyperparameter, or (2) trade off updates for different tasks according to a schedule, possibly requiring warm-start pre-training of some task (Zhang and Weiss, 2016). Even with these techniques, model parameters that perform best for one task are seldom the same as those that result in the best performance for other tasks. I believe we could improve upon these heuristics with a task-aware optimizer that adapts per-task learning rates and momentum as a function of the gradients, following the success of adaptive methods such as Adam (Kingma and Ba, 2015) for stochastic gradient descent.

To date my work on joint modeling has focused on how to learn functions that incorporate rich context across multiple tasks over model *inputs*, pushing as much information as possible regarding co-occurrence statistics between labels into e.g. contextualized token representations. Joint graphical models perform reasoning not in the space of features, but instead in the joint space of *class labels*. But exact inference in this large space is slow, and can require more training data to capture the complex co-occurrence statistics that make this modeling beneficial over single-task modeling. In response an interesting future direction is exploring techniques for approximate joint inference which perform this reasoning by embedding class labels from disparate tasks into a shared space. Such lower-dimensional approaches could result in richer models that are more accurate than those that reason over only representations of the input, while running much faster on modern hardware than traditional joint models that explicitly reason over the full joint space.

5.4 Beyond English NLP

Another generalization problem is that most NLP research published in mainstream venues is on English text, where the majority of annotation efforts have been focused. Indeed, this thesis focused on building a fast and robust NLP pipeline for English only. But this excludes most of the globe: English has only the third most native speakers in the world after Mandarin Chinese and Spanish, and considering non-native speakers, still covers only about a sixth of the world population, and particularly excludes non-native speakers of lower socioeconomic status, who are less likely to access to high-quality English language instruction. So we want techniques that generalize not only across text domains within the same language, but also across languages.

Ideally, as with modeling across tasks, we will achieve this by modeling as many different languages together in a single model, leveraging shared structure across different languages as much as possible. Indeed, there already exists a single multilingual BERT model trained on over 100 languages, as well as polyglot models for parsing (Ammar et al., 2016), SRL (Akbik and Li, 2016; Mulcaire et al., 2018) and others. At the same time, languages differ widely in their structure, at the sub-word, sentence, and even document level, and capturing these structures in a way that shares information without erasing it when needed will be an important aspect of multilingual modeling, and in particular of modeling linguistic structure alongside data-driven machine learning models such as neural networks. Since this thesis focuses only on the morphologically-poor English language, an important area for future work which applies to a more diverse group of languages will be incorporating effective sub-word representations for better e.g. morphological analysis, and exploring the extent to which explicit representations of such structures are useful or necessary for downstream tasks in light of extensive self-supervised pre-training, just as we explored the relationship between syntax and SRL in this work.

5.5 Multi-task modeling for more robust document- and corpus-level reasoning

Over-fitting in NLP models is an even bigger problem for tasks that require higher-level reasoning over larger contexts, and correspondingly larger models that require even more data, such as coreference resolution, reading comprehension and question answering. A recent analysis of a wide array of coreference systems (Moosavi and Strube, 2017), including both classic statistical NLP models as well as more recent neural network models, found that accuracy in these models degrades substantially when they are evaluated on data whose distribution differs from that on which they were trained (specifically, going from CoNLL-2012, drawn from seven domains including written and broadcast news and telephone conversations, to WikiCoref, drawn from the text of Wikipedia articles). The authors found that a substantial portion of the entities in the CoNLL-2012 test data also occur in the training data, allowing the models to perform well by over-fitting to this relatively small set of entities. Similarly, end-to-end neural network models for question answering that were supposedly surpassing human-level performance were found to be over-fitting structural regularities in the data, and could be made to fail drastically by simply modifying one sentence in the text (Jia and Liang, 2017), and models trained on the popular Stanford Natural Language Inference dataset (Bowman et al., 2015) were found to have overfit to annotation artifacts inthe data unrelated to the task of natural language inference (Gururangan et al., 2018; Poliak et al., 2018). The datasets in question have since been updated to discourage the models from learning those specific pathological behaviors (Rajpurkar et al., 2018; Williams et al., 2018), but will not necessarily stop the model from over-fitting to different, less obvious aspects of the training data. The work in Chapter 4 showed that combining a deep neural network architecture with explicit linguistic structure led to much better generalization for SRL, and a potential area for future research is to develop new techniques that will push this research forward to more complex tasks that require reasoning across entire documents, such as mention finding and coreference resolution, and ultimately across multiple documents, at the corpus level, such as reading comprehension and question answering.

As I have shown for syntactic parsing and SRL, I believe that training a single model aware of as many related NLP tasks as possible is a promising approach on this front. For example, consider the case of mention finding and coreference resolution. I hypothesize that modeling mention-finding and coreference in a single model with dependency parsing and semantic role labeling will result in more robust coreference resolution, measured by better performance out-of-domain. Providing these models with more generic syntactic and semantic information, instead of relying wholly on the word surface forms as input, could allow the model to leverage this information to better predict mentions and coreference between them when provided with unseen entities. I also hypothesize that SRL information may help with ambiguous pronominal coreference. My reasoning is based on the idea that a pronominal mention participating in a certain semantic frame may be more likely to be coreferent with certain mentions than others, and providing the model with explicit training data to this effect will help the model learn these interactions. Indeed, previous work found that adding SRL features to a coreference model improved coreference performance (Ponzetto and Strube, 2006).

Expanding multi-task models to more tasks which require incorporating information from increasingly wide context is going to require clever hierarchical modeling in order to selectively discard information that is no longer useful or relevant at certain levels. For example, it is likely not necessary to maintain a representation of the entire syntax tree for every sentence in every document in order to perform question answering, however, there is likely still information from the syntactic parse that could help with the end task. So we want a model that incorporates this information at some level and selectively allows that information to flow to higher levels as needed; hierarchical multi-task modeling allowing for adaptive amounts of information granularity at different levels, depending on how challenging the example.

Many of the most challenging ambiguities in text processing come from a lack of commonsense knowledge in our models. Commonsense knowledge can be thought of as the widest possible amount of context – that which extends beyond what is written in the text of the corpus. Developing effective techniques for extracting and incorporating this information into models for text understanding, likely via multimodel machine learning with images, audio, and even embodiment, will be paramount to building the high-quality NLP models we ultimately desire.

BIBLIOGRAPHY

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow.org*.

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 132–140.

Alan Akbik and Yunyao Li. 2016. POLYGLOT: Multilingual semantic role labeling with unified labels. In *Proceedings of ACL-2016 System Demonstrations*, pages 1–6, Berlin, Germany. Association for Computational Linguistics.

Héctor Martínez Alonso and Barbara Plank. 2017. When is multitask learning effective? semantic sequence prediction under varying data conditions. In *EACL*.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The Berkeley FrameNet project. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) – Volume 1*, pages 86–90. Association for Computational Linguistics.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010.

Marzieh Bazrafshan and Daniel Gildea. 2013. Semantic roles for string to tree machine translation. In *ACL*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Eric Bengtson and Dan Roth. 2008. Understanding the value of features for coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 294–303. Association for Computational Linguistics.

Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Brad Huang, Christopher D. Manning, Abby Vander Linden, Brittany Harding, and Peter Clark. 2014. Modeling biological processes for reading comprehension. In *EMNLP*.

Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. 1995. Bracketing guidelines for treebank ii style penn treebank project. *University of Pennsylvania*, 97:100.

Joachim Bingel and Anders Søgaard. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks. In *EACL*.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics (COLING '10)*.

Claire Bonial, Olga Babko-Malaya, Jinho D. Choi, Jena Hwang, and Martha Palmer. 2010. Propbank annotation guidelines. Technical report, Center for Computational Language and Education Research, Institute of Cognitive Science, University of Colorado at Boulder, Boulder, Colorado.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

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

Razvan Bunescu and Raymond J. Mooney. 2004. Collective information extraction with relational markov networks. In *ACL*, pages 439–446.

Vctor Campos, Brendan Jou, Xavier Gir i Nieto, Jordi Torres, and Shih-Fu Chang. 2018. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic. Association for Computational Linguistics.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the conll-2005 shared task: Semantic role labeling. In *CoNLL*.

Rich Caruana. 1993. Multitask learning: a knowledge-based source of inductive bias. In *ICML*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to search better than your teacher. In *ICML*.

Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. 1993. Equations for part-of-speech tagging. In *AAAI*, pages 695–698.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2015. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*.

Minmin Chen, Zhixiang "Eddie" Xu, Kilian Q Weinberger, Olivier Chappele, and Dor Kedem. 2012. Classifier cascade for minimizing feature evaluation cost. In *AISTATS*.

Yun-Nung Chen, William Yang Wang, and Alexander I Rudnicky. 2013. Unsupervised induction and filling of semantic slots for spoken dialogue systems using frame-semantic parsing. In *Proc. of ASRU-IEEE*.

Jason PC Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

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 *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Jinho Choi and Martha Palmer. 2011a. Getting the Most out of Transition-based Dependency Parsing. *Association for Computational Linguistics*, pages 687–692.

Jinho Choi and Martha Palmer. 2012. Fast and robust part-of-speech tagging using dynamic model selection. In *Association for Computational Linguistics*.

Jinho D. Choi and Martha Palmer. 2011b. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: short papers*, pages 687–692.

K. W. Church. 1989. A stochastic parts program and noun phrase parser for unrestricted text. In *ICASSP-89*, pages 695–698.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics.

Michael Collins. 1999. *Head-driven Statistical Models for Natural Language Parsing.* Ph.D. thesis, University of Pennsylvania.

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

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems 28 (NIPS).*

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75(3):297–325.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The stanford typed dependencies representation. In *COLING 2008 Workshop on Cross-framework and Cross-domain Parser Evaluation.*

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat. 2016. Incorporating nesterov momentum into adam. In *ICLR Workshop track.*

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *ICLR.*

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *JMLR*, 12:2121–2159.

Greg Durrett and Dan Klein. 2014. A joint model for entity analysis: Coreference, typing and linking. *Transactions of the Association for Computational Linguistics*, 2:477–490.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *NAACL*. Association for Computational Linguistics.

Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. *The Annals of Statistics*, 32(2):407–499.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345.

David Embicka and David Poeppel. 2015. Towards a computational(ist) neurobiology of language: correlational, integrated and explanatory neurolinguistics. *Language, Cognition and Neuroscience*, 30(4):357–366.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 72–78.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pages 363–370.

Nicholas FitzGerald, Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Semantic role labeling with neural network factors. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 960–970.

W. N. Francis and H. Kučera. 1964. Manual of information to accompany a standard corpus of present-day edited american english, for use with digital computers. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island.

W. Nelson Francis. 1965. A standard corpus of edited present-day american english. *College English*, 26(4):267–273.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35.

Jesús Giménez and Lluís Màrquez. 2004. Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th LREC*, Lisbon, Portugal.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012: Technical Papers*, pages 959–976.

Alexander Grubb and J. Andrew Bagnell. 2012. SpeedBoost: Anytime Prediction with Uniform Near-Optimality. In *AISTATS*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations (ICLR)*.

Tao Gui, Qi Zhang, Haoran Huang, Minlong Peng, and Xuanjing Huang. 2017. Part-of-speech tagging for twitter with adversarial neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2411–2420, Copenhagen, Denmark. Association for Computational Linguistics.

Çalar Gülçehre and Yoshua Bengio. 2016. Knowledge matters: Importance of prior information for optimization. *Journal of Machine Learning Research*, 17(8):1–32.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štepánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18. Association for Computational Linguistics.

James Hammerton. 2003. Named entity recognition with long short-term memory. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL*, pages 172–175. Association for Computational Linguistics.

Z. S. Harris. 1951. *Methods in Structural Linguistics*. University of Chicago Press, Chicago.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Conference on Empirical Methods in Natural Language Processing*.

Trevor Hastie, Jonathan Taylor, Robert Tibshirani, Guenther Walther, et al. 2007. Forward stagewise regression and the monotone lasso. *Electronic Journal of Statistics*, 1:1–29.

He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic feature selection for dependency parsing. In *EMNLP*.

He He and Jason Eisner. 2012. Cost-sensitive dynamic feature selection. In *ICML Workshop on Inferning: Interactions between Inference and Learning*.

Luheng He, Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2018. Jointly predicting predicates and arguments in neural semantic role labeling. In *ACL*.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and whats next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

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

M Honnibal and Y Goldberg. 2013. A Non-Monotonic Arc-Eager Transition System for Dependency Parsing. *CoNLL*.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 57–60.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

D. H. Hubel and T. N. Wiesel. 1962. Receptive fields, binocular vision and functional architecture in the cat's visual cortex. *J. Physiol.*, 160:106–154.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *CoRR*, abs/1707.07328.

Richard Johansson and Pierre Nugues. 2008. Dependency-based semantic role labeling of propbank. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 69–78.

Shafiq Joty, Preslav Nakov, Lluís Màrquez, and Israa Jaradat. 2017. Cross-language learning with adversarial neural networks. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 226–237. Association for Computational Linguistics.

Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. In *44th International Symposium on Computer Architecture (ISCA)*.

Jon H. Kaas, Troy A. Hackett, and Mark Jude Tramo. 1999. Auditory processing in the primate cerebral cortex. *Current Opinion in Neurobiology*, 9:164–170.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations (ICLR)*, San Diego, California, USA.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Thomas N. Kipf and Max Welling. 2017. Semisupervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Karin Kipper-Schuler. 2005. *VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon*. Ph.D. thesis, University of Pennsylvania.

Ron Kohavi and George H John. 1997. Wrappers for feature subset selection. *Artificial intelligence*, 97(1):273–324.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, Prague, Czech Republic. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL*.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Chen-Yu Lee, Saining Xie, Patrick W Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-supervised nets. *AISTATS*, 2(3):5.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *EMNLP*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *EMNLP*.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding cnns for text: non-linear, non-consecutive convolutions. *Empirical Methods in Natural Language Processing*.

Beth Levin. 1993. *English verb classes and alternations: A preliminary investigation*. University of Chicago press.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308. Association for Computational Linguistics.

Omer Levy and Yoav Goldberg. 2014b. Neural word embeddings as implicit matrix factorization. In *NIPS*.

Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint A* CCG Parsing and Semantic Role Labeling. In *EMNLP*.

Percy Liang, Hal Daumé III, and Dan Klein. 2008. Structure compilation: trading structure for features. In *Proceedings of the 25th international conference on Machine learning*, pages 592–599. ACM.

Jindřich Libovický and Jindřich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021.

Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, and Silvio Amir. 2013. Not all contexts are created equal: Better word representations with variable attention. In *EMNLP*. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015a. Two/too simple adaptations of Word2Vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, Denver, Colorado. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015b. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015c. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *EMNLP*.

Ding Liu and Daniel Gildea. 2010. Semantic role features for machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*.

Liyuan Liu, Xiang Ren, Jingbo Shang, Jian Peng, and Jiawei Han. 2018. Efficient contextualized representation: Language model pruning for sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1215–1225.

Yang Liu and Mirella Lapata. 2018. Learning structured text representations. *Transactions of the Association for Computational Linguistics*, 6:63–75.

Ben London, Bert Huang, and Lise Getoor. 2016. Stability and generalization in structured prediction. *Journal of Machine Learning Research*, 17(222):1–52.

Aurélie C Lozano, Grzegorz Swirszcz, and Naoki Abe. 2011. Group orthogonal matching pursuit for logistic regression. In *International Conference on Artificial Intelligence and Statistics*, pages 452–460.

Gang Luo, Xiaojiang Huang, Chin-Yew Lin, , and Zaiqing Nie. 2015. Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 879–888.

Xuezhe Ma, Yingkai Gaom, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. 2017. Dropout with expectation-linear regularization. In *ICLR*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, page 10641074.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, volume 30.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *12th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

Diego Marcheggiani, Anton Frolov, and Ivan Titov. 2017. A simple and accurate syntax-agnostic neural model for dependency-based semantic role labeling. In *CoNLL*.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993a. Building a large annotated corpus of English: The Penn TreeBank. *Computational Linguistics – Special issue on using large corpora: II*, 19(2):313–330.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993b. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

André Martins, Noah Smith, Pedro Aguiar, and Mário Figueiredo. 2011. Structured sparsity in structured prediction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1500–1511. Association for Computational Linguistics.

Andrew McCallum and Wei Li. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *CoNLL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Nafise Sadat Moosavi and Michael Strube. 2017. Lexical features in coreference resolution: To be used with caution. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 14–19. Association for Computational Linguistics.

Phoebe Mulcaire, Swabha Swayamdipta, and Noah A. Smith. 2018. Polyglot semantic role labeling. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 667–672, Melbourne, Australia. Association for Computational Linguistics.

Yurii Nesterov. 1983. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT*, pages 149–160, Nancy, France.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. Association for Computational Linguistics.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, volume 1, pages 351–359. Association for Computational Linguistics.

Hiroki Ouchi, Hiroyuki Shindo, and Yuji Matsumoto. 2018. A span selection model for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1630–1642, Brussels, Belgium. Association for Computational Linguistics.

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

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30 th International Conference on Machine Learning.*

Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon infused phrase embeddings for named entity resolution. In *CoNLL*.

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

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Eighth International Conference on Language Resources and Evaluation (LREC)*.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis only baselines in natural language inference. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 180–191, New Orleans, Louisiana. Association for Computational Linguistics.

Simone Paolo Ponzetto and Michael Strube. 2006. Semantic role labeling for coreference resolution. In *ACL Demonstrations*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Bj orkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2006. Towards robust linguistic analysis using ontonotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Proceedings of the Joint Conference on EMNLP and CoNLL: Shared Task*, pages 1–40.

Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Dan Jurafsky. 2005. Semantic role labeling using different syntactic views. In *Proceedings of the Association for Computational Linguistics 43rd annual meeting (ACL)*.

Vasin Punyakanok, Dan Roth, and Wen-Tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011a. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2011b. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Michael Roth and Mirella Lapata. 2016. Neural semantic role labeling with dependency path embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1192–1202.

Sascha Rothe and Hinrich Schütze. 2015. AutoExtend: Extending word embeddings to embeddings for synsets and lexemes. 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)*, pages 1793–1803, Beijing, China. Association for Computational Linguistics.

Alexander M Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics.

M. Schuster and K. Nakajima. 2012. Japanese and korean voice search.

M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 231–235.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate sequence labeling with iterated dilated convolutions. *EMNLP*.

Mihai Surdeanu, Lluís Màrquez, Xavier Carreras, and Pere R. Comas. 2007. Combination strategies for semantic role labeling. *Journal of Artificial Intelligence Research*, 29:105–151.

Charles Sutton and Andrew McCallum. 2004. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning*.

Charles Sutton and Andrew McCallum. 2005. Joint parsing and semantic role labeling. In *CoNLL*.

Swabha Swayamdipta, Sam Thomson, Chris Dyer, and Noah A. Smith. 2017. Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. In *arXiv:1706.09528*.

Grzegorz Swirszcz, Naoki Abe, and Aurelie C Lozano. 2009. Grouped orthogonal matching pursuit for variable selection and prediction. In *Advances in Neural Information Processing Systems*, pages 1150–1158.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Efficient inference and structured learning for semantic role labeling. *TACL*, 3:29–41.

Zhixing Tan, Mingxuan Wang, Jun Xie, Yidong Chen, and Xiaodong Shi. 2018. Deep semantic role labeling with self-attention. In *AAAI*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal. Association for Computational Linguistics.

Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. 2008. A global joint model for semantic role labeling. *Computational Linguistics*, 34(2):161–191.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

Kirill Trapeznikov and Venkatesh Saligrama. 2013. Supervised sequential classification under budget constraints. In *AISTATS*.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-first International Conference on Machine Learning*, page 104.

Gokhan Tur, Dilek Hakkani-Tür, and Ananlada Chotimongkol. 2005. Semi-supervised learning for spoken language understanding using semantic role labeling. In *ASRU*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *31st Conference on Neural Information Processing Systems (NIPS)*.

Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–511. IEEE.

Hai Wang, Mohit Bansal, Kevin Gimpel, and David McAllester. 2015. Machine comprehension with syntax, frames, and semantics. In *ACL*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. 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)*, pages 323–333, Beijing, China. Association for Computational Linguistics.

David Weiss and Ben Taskar. 2010. Structured prediction cascades. In *AISTATS*.

David Weiss and Ben Taskar. 2013. Learning adaptive value of information for structured prediction. In *NIPS*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

R. J. Williams and D. Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Lin Xiao. 2009. Dual Averaging Method for Regularized Stochastic Learning and Online Optimization. In *NIPS*.

Zhixiang "Eddie" Xu, Matt J Kusner, Kilian Q Weinberger, and Minmin Chen. 2013. Cost-sensitive tree of classifiers. In *ICML*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206.

Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. 2016. Multi-task cross-lingual sequence tagging from scratch. In *arXiv preprint arXiv:1603.06270*.

Fisher Yu and Vladlen Koltun. 2016. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*.

Ming Yuan and Yi Lin. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, et al. 2017. Conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015a. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28 (NIPS)*.

Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yao, Sanjeev Khudanpur, and James Glass. 2015b. Highway long short-term memory rnns for distant speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566. Association for Computational Linguistics.

Jie Zhou and Wei Xu. 2015a. End-to-end learning of semantic role labeling using recurrent neural 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)*, pages 1127–1137, Beijing, China. Association for Computational Linguistics.

Jie Zhou and Wei Xu. 2015b. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*.