Neural Networks for Linguistic Structured Prediction and Their Interpretability

Xuezhe Ma

CMU-LTI-xx-xxx

March 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

Abstract

Linguistic structured prediction, such as sequence labeling, syntactic and semantic parsing, and coreference resolution, is one of the first stages in deep language understanding and its importance has been well recognized in the natural language processing community, and has been applied to a wide range of down-stream tasks.

Most traditional high performance linguistic structured prediction models are linear statistical models, including Hidden Markov Models (HMM) and Conditional Random Fields (CRF), which rely heavily on hand-crafted features and task-specific resources. However, such task-specific knowledge is costly to develop, making structured prediction models difficult to adapt to new tasks or new domains. In the past few years, non-linear neural networks with as input distributed word representations have been broadly applied to NLP problems with great success. By utilizing distributed representations as inputs, these systems are capable of learning hidden representations directly from data instead of manually designing hand-crafted features.

Despite the impressive empirical successes of applying neural networks to linguistic structured prediction tasks, there are at least two major problems: 1) there is no a consistent architecture for, at least of components of, different structured prediction tasks that is able to be trained in a truely end-to-end setting. 2) The end-to-end training paradigm, however, comes at the expense of model interpretability: understanding the role of different parts of the deep neural network is difficult.

In this thesis, we will discuss the two of the major problems in current neural models, and attempt to provide solutions to address them. In the first part of this thesis, we introduce a consistent neural architecture for the encoding component, named BLSTM-CNNs, across different structured prediction tasks. It is a truly end-to-end model requiring no task-specific resources, feature engineering, or data pre-processing beyond pre-trained word embeddings on unlabeled corpora. Thus, our model can be easily applied to a wide range of structured prediction tasks on different languages and domains. We apply this encoding architecture to different tasks including sequence labeling and graph and transition-based dependency parsing, combined with different structured output layers, achieving state-of-the-art performance.

In the second part of this thesis, we use probing methods to investigate learning properties of deep neural networks with dependency parsing as a test bed. We first apply probes to neural dependency parsing models and demonstrate that using probes with different expressiveness leads to inconsistent observations. Based on our findings, we propose to interpret performance of probing tasks with two separate metrics, *capacity* and *accessibility*, which are associated with probe expressiveness. Specifically, *capacity* measures how much information has been encoded, while *accessibility* measures how easily the information can be detected. Then, we conduct systematic experiments to illustrate two learning properties of deep neural networks: (i) *laziness* – storing information in a way that requires minimal efforts; (ii) *targetedness* – filtering out from internal representations information that is unnecessary for the target task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Teaching machines to understand human language documents is one of the most elusive and long-standing challenges in Artificial Intelligence. Linguistic structured prediction, such as sequence labeling [81, 91, 124, 133], syntactic and semantic parsing [23, 75, 93, 97, 98, 110, 119], and coreference resolution [44, 99, 117], is one of the first stages in deep language understanding and its importance has been well recognized in the natural language processing (NLP) community.

Many problems in machine learning involve structured prediction, i.e., predicting a group of outputs that depend on each other. Many NLP systems, sentiment analysis [145], machine translation [12, 156], information extraction [7, 118, 127], word sense disambiguation [48], and low-resource languages processing [96, 112], are becoming more sophisticated, in part because of utilizing structured knowledge such as part-of-speech (POS) tags, dependency parsing trees, and entity/event coreference information. Figure 1.1 illustrates examples for three classic linguistic structured prediction tasks:

- **part-of-speech tagging:** to assign to each word in a sentence its part-of-speech (POS) tag to represent the syntax function. For example, in the sentence *I saw a girl with a telescope.* in Figure 1.1, *I* is a pronoun, *saw* is a verb is its past tense, *a* is a determiner, *girl* and *telescope* are common nouns, and *with* is a preposition.
- **named entity recognition:** to recognize the spans of named entities in a sentence. For

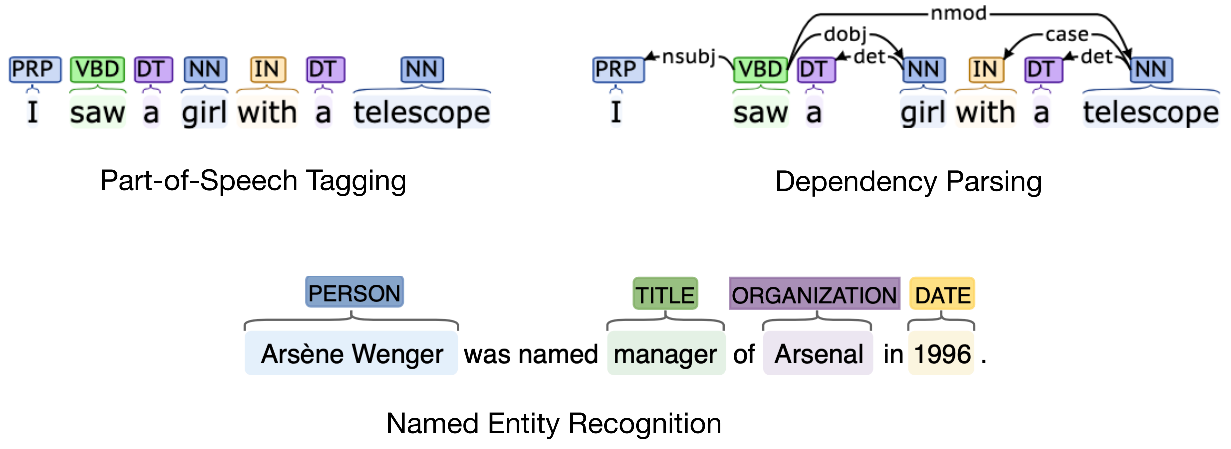


Figure 1.1: Examples for three linguistic structured prediction tasks: Part-of-speech tagging, named entity recognition and dependency parsing.

example, *Arsène Wegner* is a person's name, while *Arsenal* is an organization.

- **dependency parsing:** to analyze the syntactic dependency structure of a given sentence to represent syntactic relations between words. Taking the first sentence *I saw a girl with a telescope.* as an example again, *I* is the subject and *a girl* is the object of the verb *saw*, while *with a telescope* as a while is a prepositional phrase which describes a preposition of manner for the verb.

Most traditional high performance linguistic structured prediction models are linear statistical models, including Hidden Markov Models (HMM) and Conditional Random Fields (CRF) [91, 124, 133], which rely heavily on hand-crafted features and task-specific resources. For example, English POS taggers benefit from carefully designed word spelling features; orthographic features and external resources such as gazetteers are widely used in NER; carefully selected combinations of head and context words, and their corresponding morphological features, significantly improve the performance of dependency parsers. However, such task-specific knowledge is costly to develop [96], making structured prediction models difficult to adapt to new tasks or new domains.

Representation learning techniques based on deep neural networks [84] has re-emerged as one viable solution to this challenge, and fundamentally transformed the conventional feature engineering paradigm. Representation learning, in principle, enables automatic extraction of dense, continuous feature representations for downstream tasks via the end-to-end learning paradigm [17, 84]. The expressive power of this end-to-end learning paradigm has e led to a number of impressive empirical successes in natural language processing (NLP) and other spheres of artificial intelligence (AI), such as computer vision [78], control [116], robotics [86] and several sub-fields in machine learning [59]. In particular, deep learning has revolutionized natural language processing (NLP), with the primary advancement that words, concepts and contexts which were previously represented as a set of sparse discrete features can be represented as dense, real-valued vectors [61, 114, 128]. Compared to discrete representations which are sparse and can only attain coarse relationships, continuous representations can capture fine-grained similarities between objects. Furthermore, these continuous representations can be updated via the end-to-end representation learning paradigm to optimize the entire neural networks towards the ultimate tasks, such as machine translation [9, 144], structured prediction [31, 60] and language generation [62, 113].

Despite the impressive empirical successes of applying neural networks to linguistic structured prediction tasks, there are still some problems.

Architectures of neural networks. From the perspective of model architectures, there are two main problems in current models. First, even systems that have utilized distributed representations as inputs, however, have used these to augment, rather than replace, hand-crafted features (e.g. word spelling and capitalization patterns). Their performance drops rapidly when the models solely depend on neural embeddings. Second, there is no a consistent architecture for, at least of components of, different structured prediction tasks. For different tasks, we have to design specific neural architectures.

Interpretability of neural networks. From the perspective of model interpretability, understanding the role of different parts of the deep neural network is difficult. The end-to-end training paradigm significantly simplifies the hand-crafted feature engineering process in traditional feature-based machine learning (ML) systems. This, however, often comes at the expense of model interpretability. Unlike traditional feature-engineered NLP systems whose features, e.g. morphological properties, syntactic categories or semantic relations, are more easily understood by humans, it is more difficult to understand what happens in the internal components of an end-to-end neural network. Such deep neural models are sometimes perceived as “black-box”, hindering research efforts and limiting their utility to society [13].

In this thesis, we attempt to explore solutions to address the problems of neural networks on linguistic structured prediction, from these two aspects. Therefore, in this thesis, we explore two research directions which employ linguistic structured prediction as a key component:

- **Neural architecture:** consistent neural network architectures that not only support truly end-to-end learning of features from task-oriented (labeled) data, but also be applicable to different tasks
- **Interpretability:** learning properties of neural networks that help us understand the behaviors of the learning procedure.

In this thesis, we will focus only on linguistic structured prediction tasks on single sentence.

1.2 Thesis Outline

Following the two central themes that we just discussed, this thesis consists of two parts — PART I neural architectures and PART II interpretability.

Part I of this thesis focuses on developing neural networks for linguistic structured prediction tasks. In Chapter 2, we first visit some background of linguistic structured prediction, end-to-end learning paradigm and give an overview of the history and recent development of neural representation learning on linguistic structured prediction. In the last section of this chapter, we propose a consistent neural architecture, named BLSTM-CNNs, for the encoding component (encoder) across different structured prediction tasks. Next, by stacking different structured decoding layers on top of this encoder, we proposed deep neural models for different linguistic structured prediction tasks.

In Chapter 3, we apply BLSTM-CNNs to sequence labeling tasks, by combining a sequential CRF on top of it, to jointly decode labels for the whole sentence. With no feature engineering and data pre-processing, our model surpassed the performance of all prior models, achieving state-of-the-art performance on two classic sequence labeling tasks — part-of-speech (POS) tagging and named entity recognition (NER). This work was published as Ma and Hovy [94].

In Chapter 4 and 5, we demonstrate applications of LSTM-CNNs to graph-based and transition-based dependency parsers. For graph-based dependency parsing, we proposed the NeuroMST parser, which constructs a probabilistic structured layer on top of BLSTM-CNNs to define the conditional distribution over all dependency trees. Benefiting from the probabilistic structured output layer, we can use negative log-likelihood as the training objective, where the partition function and marginals can be computed via Kirchhoff's Matrix-Tree Theorem [137, 150]. For transition-based dependency parsing, we proposed a novel neural architecture, *stack-pointer networks* (**STACKPTR**). Unlike traditional transition-based parser with left-to-right shift-reduce actions, the STACKPTR parser performs parsing in an incremental, top-down, depth-first fashion. This architecture makes it possible to capture information from the whole sentence and all the previously derived subtrees, while maintaining a number of parsing steps linear in the sentence length. We evaluate our two parsers on 29 treebanks across 20 languages and different dependency annotation schemas, achieving state-of-the-art performance on most of them. This work is presented in Ma and Hovy [95] and Ma et al. [101].

Part II of this thesis focuses on how to interpret the learning behaviors of neural networks.

In Chapter 6, we first briefly discuss the terminological issues regarding analysis and interpretation in machine learning. Then we revisit the probing method [13, 47, 136], and apply it to investigate how part-of-speech information are memorized in neural dependency parsers.

In Chapter 7, we conduct systematic experiments to illustrate two learning properties of deep neural networks: (i) *laziness* – modules of a neural network will not actively learn information that is already learned by other modules; (ii) *targetedness* – information, if unnecessary for the end task, will be filtered out from the internal representations.

In Chapter 8, we finally conclude and discuss future work and open questions in this field.

1.3 Summary of Contributions

The contributions of this thesis are summarized as follows:

- We were among the first to research neural networks on linguistic structured prediction. In particular, we proposed three neural networks, one for sequence labeling and two for dependency parsing, all of them have demonstrated superior performance on various datasets in different languages.

- We made the effort to understand better what linguistic knowledge neural dependency parsing models have actually learned, with probing methods. We concluded that the accuracy of probes with different expressiveness does not consistently reflect the quantity of the encoded information, and further propose to measure the information encoded in representations instead using two complementary metrics.
- Finally, we pioneered the research direction of using probing methods to investigate the learning behaviors of deep neural networks. In particular, we conduct systematic experiments to illustrate two learning properties of deep neural networks: *laziness* — information, if already been encoded in some components of a neural network, will not be propagated to other components; and *targetedness* — — information that is unnecessary for the ultimate objective will be filtered out.

Part I

Neural Models for End-to-end Linguistic Structured Prediction

Chapter 2

Background

2.1 Three Tasks in Linguistic Structured Prediction

In this section, we briefly introduce three tasks in linguistic structured prediction that this thesis focuses on — two tasks in sequence labeling and one in syntactic tree parsing. These tasks are one of the first stages in deep language understanding, whose importance has been well recognized in the natural language processing (NLP) community.

2.1.1 Sequence Labeling

The task of sequence labeling is to find the best way to assign a categorical label to each token of a sequence. Two classic tasks in sequence labeling, which are also the tasks considered in this thesis, are part-of-speech (POS) tagging and named entity recognition (NER). As mentioned in Section 1.1, POS tagging is to assign to each word in a sentence its part-of-speech tag to represent its syntac function, while NER is to recognize the spans of named entities in the given sentence. For POS tagging, it is straight-froward to formulate it as a sequence labeling problem. For NER, a typical way to set this up as sequence labeling is to use the *BIO tagging schema* (as shown in Figure 2.1). Each word is labeled *B* (beginning) if it is the first word in a named entity, *I* (inside) if it is a subsequent word in a named entity, or *O* (outside) otherwise.

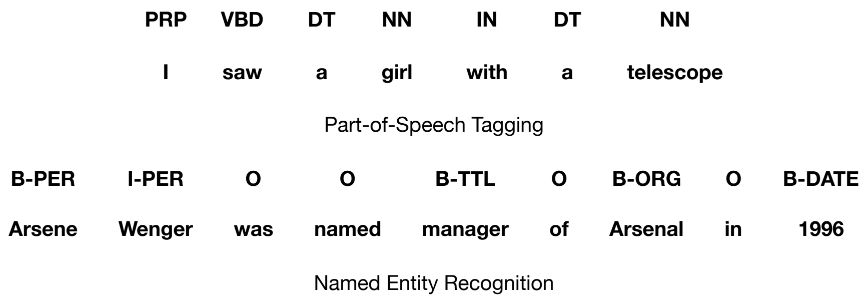

PRP VBD DT NN IN DT NN
I saw a girl with a telescope
Part-of-Speech Tagging
B-PER I-PER O O B-TTL O B-ORG O B-DATE
Arsene Wenger was named manager of Arsenal in 1996
Named Entity Recognition

Figure 2.1: Sequence labeling formulation of Part-of-speech tagging and named entity recognition. For NER, we use the BIO tagging schema.

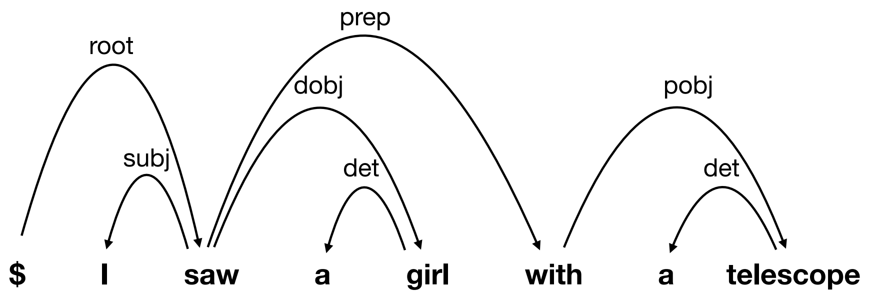


Figure 2.2: An example of dependency tree structure with dependency labels on edges.

2.1.2 Dependency Parsing

Dependency trees represent syntactic relationships between words in the sentences through labeled directed edges between head words and their dependents (modifiers). The task of dependency parsing is to automatically analyze the dependency structure for a given sentence. Figure 2.2 shows a dependency tree for the sentence *I saw a girl with a telescope.*, with the symbol $ as its root. Dependency trees are often typed with labels for each edge to represent additional syntactic information, such as *sbj* and *dobj* for verb-subject and verb-object head-modifier interactions, respectively. Sometimes, however, the dependency labels are omitted. Dependency trees are defined as labeled or unlabeled according to whether the dependency labels are included or dropped. In the remainder of this thesis, we will focus on labeled dependency parsing, i.e. jointly predicting the dependency tree structures and the dependency labels in a single parsing model.

By considering the item of crossing dependencies, dependency trees fall into two categories — projective and non-projective dependency trees. An equivalent and more convenient formulation

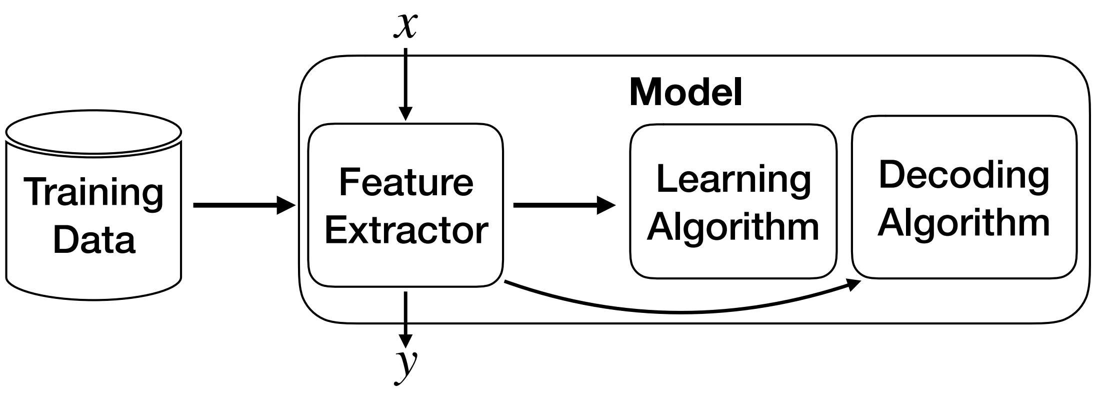


Figure 2.3: Outline of generic framework of structured prediction.

of the projectivity constrain is that if a dependency tree can be written with all words in a predefined linear order and all edges drawn on the plane without crossing edges. The example in Figure 2.2 belongs to the class of projective dependency trees. Previous studies illustrate that projective graphs are sufficient to analyze most English sentences, due to English's rigid word order [110, 158]. However, for languages with flexible word orders, non-projective graph is preferable. In this thesis, we consider non-projective dependency parsing, where we develop dependency parsers that is not restricted by the projective constraint.

2.2 General Framework of Structured Prediction

Computational methods for the linguistic structured prediction of sentences have been at the forefront of natural language processing research since its inception [22, 30, 46, 81, 132, 148]. Figure 2.3 graphically displays the framework we will assume for structured prediction.

First, a system should define a *feature extractor*, which takes as input raw sentences and outputs representations of each word that are mathematically and computationally convenient to process for machine learning algorithms. Second, the system should have a *learning algorithm* that takes the training data as input to compute the loss function. Then an optimization algorithm associated with the learning algorithm updates the parameters of the system according to the loss function. This process of producing a structured prediction model from a training set is usually called *training* or *learning*. At last, the model consists of a *decoding algorithm*, a.k.a *inference* *algorithm*, which specifies how to use the model for prediction. That is , when a new sentence is given to the model, the decoding algorithm uses the parameter specifications in the model to produce a structured output.

2.2.1 Formal Definition

In the rest of this thesis, we use the following notations: $\mathbf{x} = \{w_1, \ldots, w_n\}$ represents a generic input sentence, where w_i is the ith word. $\mathbf{y}$ represents a generic structured output, e.g. a sequence of tags or a dependency tree. $T(\mathbf{x})$ is used to denote the set of all valid structured outputs $\mathbf{y}$ for sentence $\mathbf{x}$. $\phi(w_i)$ is the feature representation of w_i, output from the feature extractor. $D = \{(\mathbf{x}_1, \mathbf{y}_1) \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ denotes our training sample, where $(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots, N$, are usually i.i.d. samples.

This thesis considers probabilistic models for structured prediction, which defines a family of conditional probability $P_\theta(\mathbf{y}|\mathbf{x})$ over all $\mathbf{y}$ given sentence $\mathbf{x}$, where $\theta \in \Theta$ is the set of parameters of this model. In the context of *maximal likelihood estimation* (MLE), parameters θ is optimized to minimizes the negative log-likelihood:

$$\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} - \log P_\theta(\mathbf{y}_i|\mathbf{x}_i) = \min_{\theta \in \Theta} \mathrm{E}_{\tilde{P}(\mathbf{y}|\mathbf{x})}[- \log P_\theta(\mathbf{y}|\mathbf{x})] \tag{2.1}$$

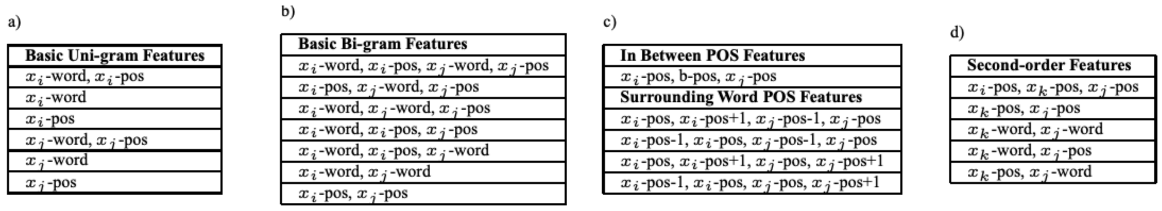
where $\tilde{P}(\mathbf{y}|\mathbf{x})$ is the empirical distribution derived from training data D. The learning algorithm is to accomplish the computation of $P_\theta(\mathbf{y}|\mathbf{x})$ for each sentence $\mathbf{x}$ and its structured output $\mathbf{y}$.

The decoding algorithm, on the other hand, is to search for the output $\mathbf{y}^*$ with the highest conditional probability:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y} \in T(\mathbf{x})} P_\theta(\mathbf{y}|\mathbf{x}) \tag{2.2}$$

Since the cardinality of $T(\mathbf{x})$ grows exponentially with the length of the sentence $\mathbf{x}$, it is infeasible to perform exhaustive search directly to sovle Eq. (2.2). Observe that both the learning and decoding algorithms rely on the formulation of the probabilistic model $P_\theta(\mathbf{y}|\mathbf{x})$, and for certain formulations, efficient algorithms exist for solving Eq. (2.1) and (2.2).

Unlike probabilistic structured prediction models, some models are trained by online learning algorithms such as averaged structured perceptron [29, 51] or margin infused relaxed algorithm [33, 34, 109]. For these models, only the algorithm for solving the decoding problem is required.



a)

Basic Uni-gram Features
x_i-word, x_i-pos
x_i-word
x_i-pos
x_j-word, x_j-pos
x_j-word
x_j-pos

b)

Basic Bi-gram Features
x_i-word, x_i-pos, x_j-word, x_j-pos
x_i-pos, x_j-word, x_j-pos
x_i-word, x_j-word, x_j-pos
x_i-word, x_i-pos, x_j-pos
x_i-word, x_i-pos, x_j-word
x_i-word, x_j-word
x_i-pos, x_j-pos

c)

In Between POS Features
x_i-pos, b-pos, x_j-pos
Surrounding Word POS Features
x_i-pos, x_i-pos+1, x_j-pos-1, x_j-pos
x_i-pos-1, x_i-pos, x_j-pos-1, x_j-pos
x_i-pos, x_i-pos+1, x_j-pos, x_j-pos+1
x_i-pos-1, x_i-pos, x_j-pos, x_j-pos+1

d)

Second-order Features
x_i-pos, x_k-pos, x_j-pos
x_k-pos, x_j-pos
x_k-word, x_j-word
x_k-word, x_j-pos
x_k-pos, x_j-word

Figure 2.4: Features used by the MST-Parser [109] for each dependency edge.

2.3 Feature Engineering vs. Representation Learning

To make task-oriented predictions, the development of machine learning systems heavily rely on extracting abstract informative features from real-world data that are represented in raw digital formats. For example, syntactic, semantic and contextual information are essentially important for a wide range of NLP tasks such as information extraction (IE). Text documents, however, are stored by individual tokens of words or characters, upon which these abstract information is hard to represent. Similar scenarios happen in image data — digital images are made up of pixels, from which it is difficult to extract abstract features such as edge or shape that are crucial for tasks, for instance, image classification.

2.3.1 Feature Engineering

In the literature of linguistic structured prediction, most traditional high performance structured prediction models extract instructive features using the hand-crafted feature-engineering process, which mainly relies on task-specific expertise and heuristically designed hand-crafted features, in an iterative feature-selection process. For example, English POS taggers benefit from carefully designed word spelling features; orthographic features and external resources such as gazetteers are widely used in NER. Figure 2.4 lists the concrete features used in the MST-Parser [109] to represent a single dependency edge $w_i \rightarrow w_j$. Table (a) and (b) show the basic set of features, which are over head-modifier paris in the tree. These features provide back-off from very specific features over words and part-of-speech (POS) tags to less sparse features over just POS tags. These features are added for both the entire words as well as the 5-gram prefix if the word is longer than 5 characters [110]. In addition to the basic set of features, McDonald and Pereira

[109] added two more types of features, listed in Table (c). The first new feature class recognizes word types that occur between the head and modifier words in an attachment decision and the second class of additional features represents the local context of the attachment, that is, the words before and after the head-modifier pair. All these features are carefully designed with linguistic knowledge and selected using the iterative feature-selection process. However, there are two problems with that brute-force methodology: 1) the combinatorial nature of empirical feature selection process makes it expensive to hand-craft features; 2) The development of these features is commonly task-, domain-, or even language-specific, preventing it from adapting to new tasks or domains.

2.3.2 End-to-end Representation Learning

Deep neural models, on the other hand, are able to automatically extract salient features for downstream tasks via the end-to-end training paradigm. The feature representation $\phi(w_i)$ is no longer binary vectors, but continuous vectors output from neural networks.

Formally, we use $\phi_{\theta'}(\mathbf{x})$ to denote the neural representation of $\mathbf{x}$ output from a neural network $\mathbf{M}_{\theta'}$, where θ' are the parameters of the neural network $\mathbf{M}$. So in deep learning models, the model parameters are the union of the parameters of the machine learning model and the neural network: $\theta \cup \theta'$. During model training the two sets of parameters are updated simultaneously to optimize the loss function in an end-to-end learning paradigm:

$$\min_{\theta,\theta' \in \Theta} \frac{1}{N} \sum_{i=1}^{N} -\log P_\theta(\mathbf{y}_i | \phi_{\theta'}(\mathbf{x}_i))$$

In the rest of this thesis, we use θ to denote the set of all the parameters of a deep learning model, when there is no ambiguity.

The end-to-end training paradigm significantly simplifies the hand-crafted feature engineering process in traditional feature-based machine learning systems, while giving the neural models flexibility to be optimized towards the ultimate tasks.

2.3.3 A Brief History of Representation Learning on Linguistic Structured Prediction

In the past few years, deep neural networks, together with end-to-end learning paradigm, have been applied to a wide range of NLP tasks with great successes.

Sequence labeling. Collobert et al. [31] proposed a simple but effective feed-forward neutral network that independently classifies labels for each word by using contexts within a window with fixed size. Recently, recurrent neural networks (RNN) [58], together with its variants such as long-short term memory (LSTM) [53, 67] and gated recurrent unit (GRU) [27], have shown great success in modeling sequential data. Several RNN-based neural network models have been proposed to solve sequence labeling tasks like speech recognition [61], POS tagging [69, 80, 134] and NER [26, 40, 68, 82, 126], achieving competitive performance against traditional models.

Dependency Parsing. For graph-based dependency parsing, Kiperwasser and Goldberg [74] and Wang and Chang [152] replaced the linear scoring function of each arc in traditional models with neural networks and used a margin-based objective [110] for model training. Kiperwasser and Goldberg [74]proposed a graph-based dependency parser which uses BLSTM for word-level representations. Wang and Chang [152] used a similar model with a way to learn sentence segment embedding based on an extra forward LSTM network. Both of these two parsers trained the parsing models by optimizing margin-based objectives. Zhang et al. [162] and Dozat and Manning [43] formalized dependency parsing as independently selecting the head of each word with cross-entropy objective, without the guarantee of a general non-projective tree structure output.

For transition-based dependency parsing, neural continuous states have been explored, in which the transition state is embedded as a neural continuous vector. Chen and Manning [23] proposed to use feed-forward neural networks to model the transition states, and Stenetorp [141] adopted recursive neural networks. Weiss et al. [153] presented structured perceptron training for neural network transition-based dependency parsing. Andor et al. [6] proposed a globally normalized transition model to replace the locally normalized classifier. Dyer et al. [45] introduced transtion-based parser using Stack LSTMs whose continuous-state embeddings were constructed using LSTM recurrent neural networks which are capable of learning representations of the parser state that are sensitive to the complete contents of the parser's state. Ballesteros et al. [10] improved the Stack-LSTM parser by replacing word representations with representations constructed from the orthographic representations of the words, and Ballesteros et al. [11] adapted the Stack-LSTM parser to support training-with-exploration procedure using dynamic oracles.

2.4 BLSTM-CNNs

From the perspective of neural network architectures, there are two main problems in current neural models for linguistic structured prediction. First, even systems that have utilized distributed representations as inputs, however, have used these to augment, rather than replace, hand-crafted features (e.g. word spelling and capitalization patterns). Their performance drops rapidly when the models solely depend on neural embeddings. Second, there is no a consistent architecture for, at least of components of, different structured prediction tasks. For different tasks, we have to design specific neural architectures.

In this section, we propose a consistent neural architecture for the encoding component across different structured prediction tasks. This neural architecture is named BLSTM-CNNs. Specificiially, we first use convolutional neural networks (CNNs) [83] to encode character-level information of a word into its character-level representation. Then we combine character- and word-level representations and feed them into bi-directional LSTM (BLSTM) to model context information of each word. It is a truly end-to-end model requiring no task-specific resources, feature engineering, or data pre-processing beyond pre-trained word embeddings on unlabeled corpora. Thus, our model can be easily applied to a wide range of structured prediction tasks on different languages and domains (see Chanper 3 and 4).

In this section, we describe the components (layers) of our neural network architecture one-by-one from bottom to top.

2.4.1 CNN for Character-level Representation

Previous studies [26, 134] have shown that CNN is an effective approach to extract morphological information (like the prefix or suffix of a word) from characters of words and encode it into neural representations. Figure 2.5 shows the CNN we use to extract character-level representation of a given word.

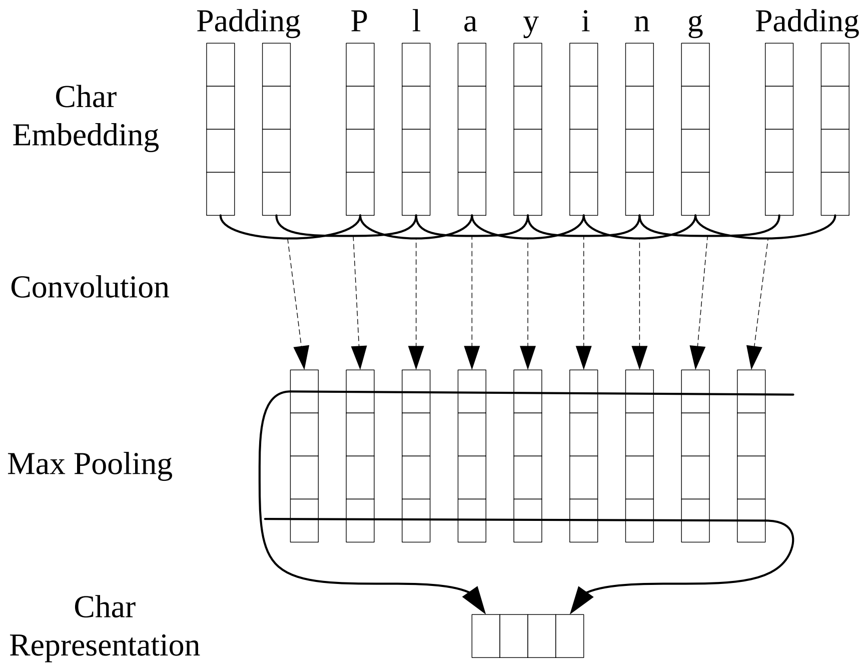


Figure 2.5: The convolution neural network for extracting character-level representations of words. Dashed arrows indicate a dropout layer applied before character embeddings are input to CNN.

The CNN is similar to the one in Chiu and Nichols [26], except that we use only character embeddings as the inputs to CNN, without character type features. A dropout layer [140] is applied before character embeddings are input to CNN.

2.4.2 Bi-directional LSTM

LSTM Unit

Recurrent neural networks (RNNs) are a powerful family of connectionist models that capture time dynamics via cycles in the graph, and are capable of dealing with variable-length sequence input. It uses a recurrent hidden state whose activation is dependent on that of the one immediate before. Though, in theory, RNNs are capable to capturing long-distance dependencies, in practice, they fail due to the gradient vanishing/exploding problems [16, 122].

Due to the vanishing gradient problem, conventional RNNs is found difficult to be trained to exploit long-range dependencies. In order to mitigate this weak point in conventional RNNs, specially designed activation functions have been introduced. LSTMs [67] are variants of RNNs designed to cope with these gradient vanishing problems. Basically, a LSTM unit is composed of three multiplicative gates which control the proportions of information to forget and to pass on to the next time step. Figure 2.6 gives the basic structure of an LSTM unit.

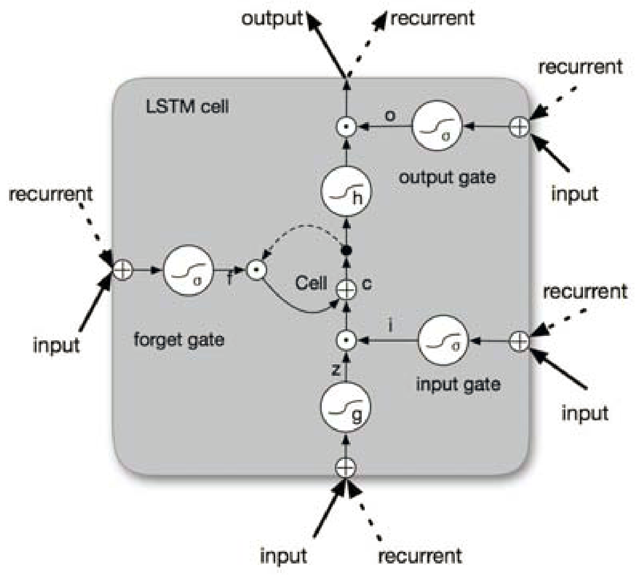


Figure 2.6: Schematic of LSTM unit.

Formally, the formulas to update an LSTM unit at time t are:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\boldsymbol{W}_i \mathbf{h}_{t-1} + \boldsymbol{U}_i \mathbf{x}_t + \boldsymbol{b}_i) \\
\mathbf{f}_t &= \sigma(\boldsymbol{W}_f \mathbf{h}_{t-1} + \boldsymbol{U}_f \mathbf{x}_t + \boldsymbol{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\boldsymbol{W}_c \mathbf{h}_{t-1} + \boldsymbol{U}_c \mathbf{x}_t + \boldsymbol{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{o}_t &= \sigma(\boldsymbol{W}_o \mathbf{h}_{t-1} + \boldsymbol{U}_o \mathbf{x}_t + \boldsymbol{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}$$

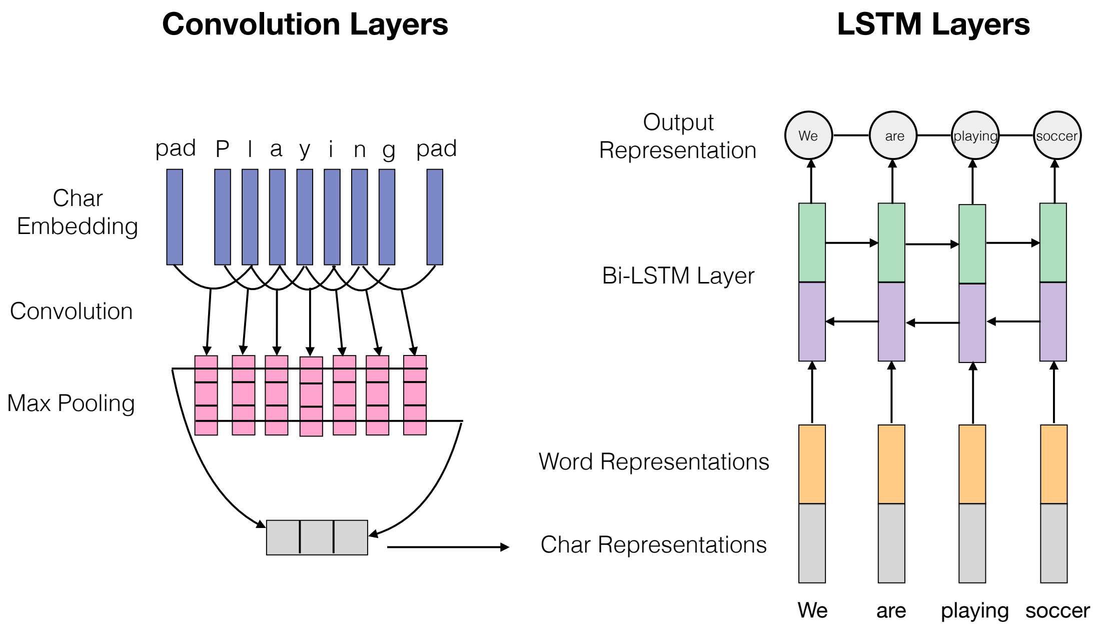


Figure 2.7: The main architecture of our encoding neural network. The character representation for each word is computed by the CNN in Figure 2.5. Then the character representation vector is concatenated with the word embedding before feeding into the BLSTM network. Dashed arrows indicate dropout layers applied on both the input and output vectors of BLSTM.

where σ is the element-wise sigmoid function and $\odot$ is the element-wise product. $\mathbf{x}_t$ is the input vector (e.g. word embedding) at time t, and $\mathbf{h}_t$ is the hidden state (also called output) vector storing all the useful information at (and before) time t. $\boldsymbol{U}_i, \boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_o$ denote the weight matrices of different gates for input $\mathbf{x}_t$, and $\boldsymbol{W}_i, \boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_o$ are the weight matrices for hidden state $\mathbf{h}_t$. $\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_c, \boldsymbol{b}_o$ denote the bias vectors.

LSTM uses input and output gates to control the flow of information through the cell. The input gate should be kept sufficiently active to allow the signals in. Same rule applies to the output gate. The forget gate is used to reset the cell's own state. In Gers et al. [54], peephole connections are usually used to connect gates to the cell in tasks requiring precise timing and counting of the internal states. It should be noted that we do not include peephole connections [54] in the our LSTM formulation because the precise timing does not seem to be required.

BLSTM

Another weak point of conventional RNNs is their utilization of only previous context with no exploitation of future context. While for many linguistic structured prediction tasks it is beneficial to have access to both past (left) and future (right) contexts. However, the LSTM's hidden state $\mathbf{h}_t$ takes information only from past, knowing nothing about the future. An elegant solution whose effectiveness has been proven by previous work [45] is bi-directional LSTM (BLSTM). The basic idea is to present each sequence forwards and backwards to two separate hidden states to capture past and future information, respectively. Then the two hidden states are concatenated to form the final output.

2.4.3 BLSTM-CNNs Encoding Architecture

For each word, the character-level representation is computed by the CNN in Figure 2.5 with character embeddings as inputs. Then the character-level representation vector is concatenated with the word embedding vector to feed into the BLSTM network. Figure 2.7 illustrates the architecture of our network in detail.

Chapter 3

Sequence Labeling via BLSTM-CNNs-CRF

This chapter describes the BLSTM-CNNs-CRF model for linguistic sequence labeling.

3.1 Neural CRF Model

Following the notations defined in Section 2.2, we use $\mathbf{x} = \{w_1, \cdots, w_n\}$ to represent a generic input sequence where w_i is the input vector of the ith word. $\boldsymbol{y} = \{y_1, \cdots, y_n\}$ represents a generic sequence of labels for $\mathbf{x}$. $T(\mathbf{x})$ denotes the set of possible label sequences for $\mathbf{x}$.

A simple and straight-forward solution to model the conditional probability $P_\theta(\mathbf{y}|\mathbf{x})$ over all $\mathbf{y}$ is to assume that each token of label sequence is independent given the input sentence $\mathbf{x}$:

$$P_\theta(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P_\theta(y_i|\mathbf{x})$$

For sequence labeling (or general structured prediction) tasks, however, it is beneficial to consider the correlations between labels in neighborhoods and jointly decode the best chain of labels for a given input sentence. For example, in POS tagging an adjective is more likely to be followed by a noun than a verb, and in NER with standard BIO2 annotation [148] I-ORG cannot follow I-PER. Therefore, we model label sequence jointly using a conditional random field (CRF) [81], instead of decoding each label independently.

3.1.1 Conditional Random Fields (CRF)

The probabilistic model for sequence CRF [81] defines a family of conditional probability $p(\boldsymbol{y}|\mathbf{x};\mathbf{W},\mathbf{b})$ over all possible label sequences $\boldsymbol{y}$ given $\mathbf{x}$ with the following form:

$$P_\theta(\boldsymbol{y}|\mathbf{x}) = \frac{\prod\limits_{i=1}^{n} \psi_i(y_{i-1}, y_i, \mathbf{x})}{\sum\limits_{y' \in T(\mathbf{x})} \prod\limits_{i=1}^{n} \psi_i(y'_{i-1}, y'_i, \mathbf{x})}$$

where $\psi_i(y', y, \mathbf{x}) = \exp(\mathbf{W}^T_{y',y}\phi(w_i) + \mathbf{b}_{y',y})$ are potential functions. $\phi(w_i)$ is the feature representation of word w_i. $\mathbf{W}^T_{y',y}$ and $\mathbf{b}_{y',y}$ are the weight vector and bias corresponding to label pair (y', y), respectively. They are learnable parameters of the model, $\mathbf{W}^T_{y',y}, \mathbf{b}_{y',y} \in \theta$.

For a sequence CRF model (only interactions between two successive labels are considered), training and decoding can be solved efficiently by adopting the Viterbi algorithm [50, 81].

3.1.2 BLSTM-CNNs-CRF

To construct our neural network model for sequence labeling, we feed the output vectors of BLSTM-CNNs into a CRF layer. Figure 3.1 illustrates the architecture of our network in detail.

For each word, the character-level representation is computed by the CNN in Figure 2.5 with character embeddings as inputs. Then the character-level representation vector is concatenated with the word embedding vector to feed into the BLSTM network. Finally, the output vectors of BLSTM ($\phi(w_i), i = 1, \ldots, n$) are fed to the CRF layer to jointly decode the best label sequence. As shown in Figure 3.1, dropout layers are applied on both the input and output vectors of BLSTM. Experimental results show that using dropout significantly improve the performance of our model (see Section 3.4.4 for details).

3.2 Network Training

In this section, we provide details about training the neural network. We implement the neural network using the Theano library [18]. The computations for a single model are run on a GeForce GTX TITAN X GPU. Using the settings discussed in this section, the model training requires about 12 hours for POS tagging and 8 hours for NER.

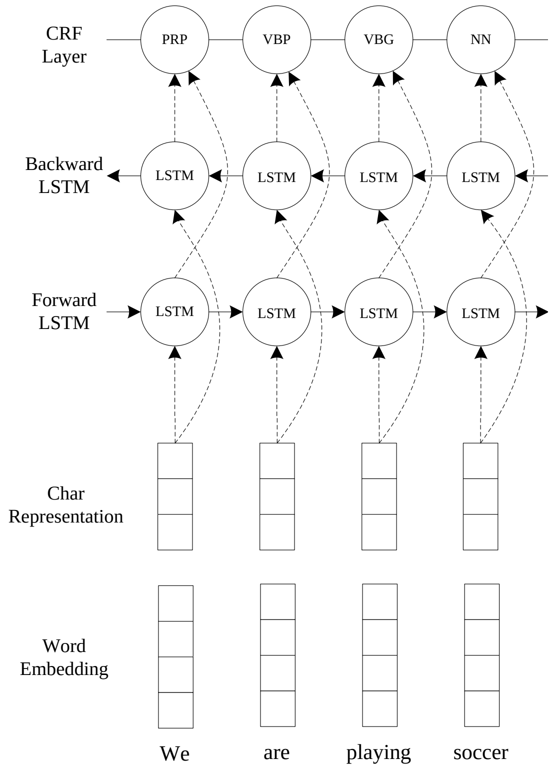


Figure 3.1: The main architecture of our neural network. The character representation for each word is computed by the CNN in Figure 2.5. Then the character representation vector is concatenated with the word embedding before feeding into the BLSTM network. Dashed arrows indicate dropout layers applied on both the input and output vectors of BLSTM.

3.2.1 Parameter Initialization

Word Embeddings. We use Stanford's publicly available GloVe 100-dimensional embeddings[1] trained on 6 billion words from Wikipedia and web text [128]

We also run experiments on two other sets of published embeddings, namely Senna 50-dimensional embeddings[2] trained on Wikipedia and Reuters RCV-1 corpus [31], and Google's Word2Vec 300-dimensional embeddings[3] trained on 100 billion words from Google News [114]. To test the effectiveness of pretrained word embeddings, we experimented with randomly initialized embeddings with 100 dimensions, where embeddings are uniformly sampled from range

[1]`http://nlp.stanford.edu/projects/glove/`
[2]`http://ronan.collobert.com/senna/`
[3]`https://code.google.com/archive/p/word2vec/`

$[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$ where dim is the dimension of embeddings [64]. The performance of different word embeddings is discussed in Section 3.4.3.

Character Embeddings. Character embeddings are initialized with uniform samples from $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$, where we set $dim = 30$.

Weight Matrices and Bias Vectors. Matrix parameters are randomly initialized with uniform samples from $[-\sqrt{\frac{6}{r+c}}, +\sqrt{\frac{6}{r+c}}]$, where r and c are the number of of rows and columns in the structure [57]. Bias vectors are initialized to zero, except the bias $\mathbf{b}_f$ for the forget gate in LSTM , which is initialized to 1.0 [70].

3.2.2 Optimization Algorithm

Parameter optimization is performed with mini-batch stochastic gradient descent (SGD) with batch size 10 and momentum 0.9. We choose an initial learning rate of η_0 ($\eta_0 = 0.01$ for POS tagging, and 0.015 for NER, see Section 3.2.3.), and the learning rate is updated on each epoch of training as $\eta_t = \eta_0/(1 + \rho t)$, with decay rate $\rho = 0.05$ and t is the number of epoch completed. To reduce the effects of "gradient exploding", we use a gradient clipping of 5.0 [122]. We explored other more sophisticated optimization algorithms such as AdaDelta [159], Adam [73] or RMSProp [37], but none of them meaningfully improve upon SGD with momentum and gradient clipping in our preliminary experiments.

Early Stopping. We use early stopping [55, 61] based on performance on validation sets. The "best" parameters appear at around 50 epochs, according to our experiments.

Fine Tuning. For each of the embeddings, we fine-tune initial embeddings, modifying them during gradient updates of the neural network model by back-propagating gradients. The effectiveness of this method has been previously explored in sequential and structured prediction problems [31, 125].

Layer	Hyper-parameter	POS	NER
CNN	window size	3	3
	number of filters	30	30
LSTM	state size	200	200
	initial state	0.0	0.0
	peepholes	no	no
Dropout	dropout rate	0.5	0.5
	batch size	10	10
	initial learning rate	0.01	0.015
	decay rate	0.05	0.05
	gradient clipping	5.0	5.0

Table 3.1: Hyper-parameters for all experiments.

Dropout Training. To mitigate overfitting, we apply the dropout method [140] to regularize our model. As shown in Figure 2.5 and 3.1, we apply dropout on character embeddings before inputting to CNN, and on both the input and output vectors of BLSTM. We fix dropout rate at 0.5 for all dropout layers through all the experiments. We obtain significant improvements on model performance after using dropout (see Section 3.4.4).

3.2.3 Tuning Hyper-Parameters

Table 3.1 summarizes the chosen hyper-parameters for all experiments. We tune the hyper-parameters on the development sets by random search. Due to time constrains it is infeasible to do a random search across the full hyper-parameter space. Thus, for the tasks of POS tagging and NER we try to share as many hyper-parameters as possible. Note that the final hyper-parameters for these two tasks are almost the same, except the initial learning rate. We set the state size of LSTM to 200. Tuning this parameter did not significantly impact the performance of our model. For CNN, we use 30 filters with window length 3.

Dataset		WSJ	CoNLL2003
Train	SENT	38,219	14,987
	TOKEN	912,344	204,567
Dev	SENT	5,527	3,466
	TOKEN	131,768	51,578
Test	SENT	5,462	3,684
	TOKEN	129,654	46,666

Table 3.2: Corpora statistics. SENT and TOKEN refer to the number of sentences and tokens.

3.3 Experiment Setup

3.3.1 Data Sets

As mentioned before, we evaluate our neural network model on two sequence labeling tasks: POS tagging and NER.

POS Tagging. For English POS tagging, we use the Wall Street Journal (WSJ) portion of Penn Treebank (PTB) [104], which contains 45 different POS tags. In order to compare with previous work, we adopt the standard splits — section 0–18 as training data, section 19–21 as development data and section 22–24 as test data [103, 138].

NER. For NER, We perform experiments on the English data from CoNLL 2003 shared task [147]. This data set contains four different types of named entities: *PERSON, LOCATION, ORGANIZATION*, and *MISC*. We use the BIOES tagging scheme instead of standard BIO2, as previous studies have reported meaningful improvement with this scheme [36, 82, 133].

The corpora statistics are shown in Table 3.2. We did not perform any pre-processing for data sets, leaving our system truly end-to-end.

	POS		NER					
	Dev	Test	Dev			Test		
Model	Acc.	Acc.	Prec.	Recall	F1	Prec.	Recall	F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BRNN-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21

Table 3.3: Performance of our model on both the development and test sets of the two tasks, together with three baseline systems.

3.4 Experimental Results

3.4.1 Main Results

We first run experiments to dissect the effectiveness of each component (layer) of our neural network architecture by ablation studies. We compare the performance with three baseline systems — BRNN, the bi-direction RNN; BLSTM, the bi-direction LSTM, and BLSTM-CNNs, the combination of BLSTM with CNN to model character-level information. All these models are run using Stanford's GloVe 100 dimensional word embeddings and the same hyper-parameters as shown in Table 3.1. According to the results shown in Table 3.3, BLSTM obtains better performance than BRNN on all evaluation metrics of both the two tasks. BLSTM-CNN models significantly outperform the BLSTM model, showing that character-level representations are important for linguistic sequence labeling tasks. This is consistent with results reported by previous work [26, 134]. Finally, by adding CRF layer for joint decoding we achieve significant improvements over BLSTM-CNN models for both POS tagging and NER on all metrics. This demonstrates that jointly decoding label sequences can significantly benefit the final performance of neural network models.

Model	Acc.
Giménez and Màrquez [56]	97.16
Toutanova et al. [149]	97.27
Manning [103]	97.28
Collobert et al. [31]‡	97.29
Santos and Zadrozny [134]‡	97.32
Shen et al. [135]	97.33
Sun [142]	97.36
Søgaard [138]	97.50
This paper	**97.55**

Model	F1
Chieu and Ng [25]	88.31
Florian et al. [49]	88.76
Ando and Zhang [5]	89.31
Collobert et al. [31]‡	89.59
Huang et al. [69]‡	90.10
Chiu and Nichols [26]‡	90.77
Ratinov and Roth [133]	90.80
Lin and Wu [87]	90.90
Passos et al. [124]	90.90
Lample et al. [82]‡	90.94
Luo et al. [91]	91.20
This paper	**91.21**

Table 3.4: Left: POS tagging accuracy of our model on test data from WSJ proportion of PTB, together with top-performance systems. Right: NER F1 score of our model on test data set from CoNLL-2003. For the purpose of comparison, we also list F1 scores of previous top-performance systems. The neural network based models are marked with ‡.

3.4.2 Comparison with Previous Work

POS Tagging. Table 3.4 (left) illustrates the results of our model for POS tagging, together with seven previous top-performance systems for comparison. Our model significantly outperform Senna [31], which is a feed-forward neural network model using capitalization and discrete suffix features, and data pre-processing. Moreover, our model achieves 0.23% improvements on accuracy over the "CharWNN" [134], which is a neural network model based on Senna and also uses CNNs to model character-level representations. This demonstrates the effectiveness of BLSTM for modeling sequential data and the importance of joint decoding with structured prediction model.

Comparing with traditional statistical models, our system achieves state-of-the-art accuracy, obtaining 0.05% improvement over the previously best reported results by Søgaard [138]. It should be noted that Huang et al. [69] also evaluated their BLSTM-CRF model for POS tagging on WSJ corpus. But they used a different splitting of the training/dev/test data sets. Thus, their results are not directly comparable with ours.

NER. Table 3.4 (right) shows the F1 scores of previous models for NER on the test data set from CoNLL-2003 shared task. For the purpose of comparison, we list their results together with ours. Similar to the observations of POS tagging, our model achieves significant improvements over Senna and the other three neural models, namely the LSTM-CRF proposed by Huang et al. [69], LSTM-CNNs proposed by Chiu and Nichols [26], and the LSTM-CRF by Lample et al. [82]. Huang et al. [69] utilized discrete spelling, POS and context features, Chiu and Nichols [26] used character-type, capitalization, and lexicon features, and all the three model used some task-specific data pre-processing, while our model does not require any carefully designed features or data pre-processing. We have to point out that the result (90.77%) reported by Chiu and Nichols [26] is incomparable with ours, because their final model was trained on the combination of the training and development data sets[4].

To our knowledge, the previous best F1 score (91.20)[5] reported on CoNLL 2003 data set is by the joint NER and entity linking model [91]. This model used many hand-crafted features including stemming and spelling features, POS and chunks tags, WordNet clusters, Brown Clusters, as well as external knowledge bases such as Freebase and Wikipedia. Our end-to-end model slightly improves this model by 0.01%, yielding a state-of-the-art performance.

3.4.3 Word Embeddings

As mentioned in Section 3.2.1, in order to test the importance of pretrained word embeddings, we performed experiments with different sets of publicly published word embeddings, as well as a random sampling method, to initialize our model. Table 3.5 gives the performance of three different word embeddings, as well as the randomly sampled one. According to the results in

[4]We run experiments using the same setting and get 91.37% F1 score.

[5]Numbers are taken from the Table 3 of the original paper [91]. While there is clearly inconsistency among the precision (91.5%), recall (91.4%) and F1 scores (91.2%), it is unclear in which way they are incorrect.

Embedding	Dimension	POS	NER
Random	100	97.13	80.76
Senna	50	97.44	90.28
Word2Vec	300	97.40	84.91
GloVe	100	**97.55**	**91.21**

Table 3.5: Results with different choices of word embeddings on the two tasks (accuracy for POS tagging and F1 for NER).

	POS			**NER**		
	Train	**Dev**	**Test**	**Train**	**Dev**	**Test**
No	98.46	97.06	97.11	99.97	93.51	89.25
Yes	97.86	97.46	97.55	99.63	94.74	91.21

Table 3.6: Results with and without dropout on two tasks.

Table 3.5, models using pretrained word embeddings obtain a significant improvement as opposed to the ones using random embeddings. Comparing the two tasks, NER relies more heavily on pretrained embeddings than POS tagging. This is consistent with results reported by previous work [26, 31, 69].

For different pretrained embeddings, Stanford's GloVe 100 dimensional embeddings achieve best results on both tasks, about 0.1% better on POS accuracy and 0.9% better on NER F1 score than the Senna 50 dimensional one. This is different from the results reported by Chiu and Nichols [26], where Senna achieved slightly better performance on NER than other embeddings. Google's Word2Vec 300 dimensional embeddings obtain similar performance with Senna on POS tagging, still slightly behind GloVe. But for NER, the performance on Word2Vec is far behind GloVe and Senna. One possible reason that Word2Vec is not as good as the other two embeddings on NER is because of vocabulary mismatch — Word2Vec embeddings were trained in case-sensitive manner, excluding many common symbols such as punctuations and digits. Since we do not use any data pre-processing to deal with such common symbols or rare words, it might be an issue for using Word2Vec.

	POS		**NER**	
	Dev	Test	Dev	Test
IV	127,247	125,826	4,616	3,773
OOTV	2,960	2,412	1,087	1,597
OOEV	659	588	44	8
OOBV	902	828	195	270

Table 3.7: Statistics of the partition on each corpus. It lists the number of tokens of each subset for POS tagging and the number of entities for NER.

3.4.4 Effect of Dropout

Table 3.6 compares the results with and without dropout layers for each data set. All other hyper-parameters remain the same as in Table 3.1. We observe a essential improvement for both the two tasks. It demonstrates the effectiveness of dropout in reducing overfitting.

3.4.5 OOV Error Analysis

To better understand the behavior of our model, we perform error analysis on Out-of-Vocabulary words (OOV). Specifically, we partition each data set into four subsets — in-vocabulary words (IV), out-of-training-vocabulary words (OOTV), out-of-embedding-vocabulary words (OOEV) and out-of-both-vocabulary words (OOBV). A word is considered IV if it appears in both the training and embedding vocabulary, while OOBV if neither. OOTV words are the ones do not appear in training set but in embedding vocabulary, while OOEV are the ones do not appear in embedding vocabulary but in training set. For NER, an entity is considered as OOBV if there exists at lease one word not in training set and at least one word not in embedding vocabulary, and the other three subsets can be done in similar manner. Table 3.7 informs the statistics of the partition on each corpus. The embedding we used is Stanford's GloVe with dimension 100, the same as Section 3.4.1.

Table 3.8 illustrates the performance of our model on different subsets of words, together with the baseline LSTM-CNN model for comparison. The largest improvements appear on the OOBV

	POS							
	Dev				Test			
	IV	OOTV	OOEV	OOBV	IV	OOTV	OOEV	OOBV
LSTM-CNN	97.57	**93.75**	90.29	80.27	97.55	**93.45**	90.14	80.07
LSTM-CNN-CRF	**97.68**	93.65	**91.05**	**82.71**	**97.77**	93.16	**90.65**	**82.49**
	NER							
	Dev				Test			
	IV	OOTV	OOEV	OOBV	IV	OOTV	OOEV	OOBV
LSTM-CNN	94.83	87.28	96.55	82.90	90.07	89.45	100.00	78.44
LSTM-CNN-CRF	**96.49**	**88.63**	**97.67**	**86.91**	**92.14**	**90.73**	100.00	**80.60**

Table 3.8: Comparison of performance on different subsets of words.

subsets of both the two corpora. This demonstrates that by adding CRF for joint decoding, our model is more powerful on words that are out of both the training and embedding sets.

3.5 Discussions

In this chapter, we equipped the LSTM-CNNs encoding architecture with a CRF output layer for sequence labeling task. The evaluation results show that our truly end-to-end BLSTM-CNNs-CRF model achieved state-of-the-art performance on two linguistic sequence labeling tasks, comparing with previous state-of-the-art systems. The ablation studies analyzed the impact of each components of the model, indicating that the improvements come from both the encoding architecture and the CRF decoding mechanism.

There are several potential directions for future work. First, our model can be further improved by exploring multi-task learning approaches to combine more useful and correlated information. For example, we can jointly train a neural network model with both the POS and NER tags to improve the intermediate representations learned in our network. Another interesting direction is to apply our model to data from other domains such as social media (Twitter and Weibo). Since our model does not require any domain- or task-specific knowledge, it might be effortless to apply it to these domains. Last but not the least, combined with other decoding layers, the BLSTM-CNNs encoding architecture can be wildly applied to other NLP tasks. In the following two chapters, we describe two neural dependency parsing models both utilizing the same encoding architecture.

Chapter 4

Neural Networks for Dependency Parsing

This chapter describes the neural networks for dependency parsing. We first review the two dominant approaches for dependency parsing — graph-based and transition-based dependency parsing models (Section 4.1). Then we introduce our NeuroMST parser for graph-based approach in Section 4.2 and leave the Stack-Pointer parser for transition-based approach in next chapter.

4.1 Two Dominant Models for Dependency Parsing

There are two dominant approaches to dependency parsing [21, 120]: local and greedy *transition-based* algorithms [23, 119, 158, 164], and the globally optimized *graph-based* algorithms [46, 75, 110, 111]. In this section, we will briefly review these two parsing models.

4.1.1 Graph-based Dependency Parsing

Formally, for a given sentence $\mathbf{x} = \{w_1, \ldots, w_n\}$, the complete dependency graph $G(\mathbf{x}) =< V(\mathbf{x}), E(\mathbf{x}) >$ represent words and their relationship to syntactic modifiers using directed edges. $V(\mathbf{x})$ and $E(\mathbf{x})$ are the sets of vertex and edges of the graph $G(\mathbf{x})$. A valid dependency tree $\mathbf{y}$ of $\mathbf{x}$ can be seen as a subgraph of $G(\mathbf{x})$ that build up a tree structure. We write $(w_i \rightarrow w_j) \in \mathbf{y}$ if there is a dependency in $\mathbf{y}$ from word w_i to word w_j. Graph-based dependency parsers learn scoring functions for parse trees and perform exhaustive search over all possible trees for a sentence to find the globally highest scoring tree.

Edge-Factored Parsing Model. A common method is to factorize the score of a dependency tree as the sum of the scores of all edges in the tree:

$$\psi(\mathbf{x}, \mathbf{y}; \theta) = \sum_{(w_h, w_m) \in \mathbf{y}} \psi(w_h, w_m; \theta)$$

where θ is the parameter and $\psi(\mathbf{x}, \mathbf{y}; \theta)$ is the score function of the parse tree $\mathbf{y}$, which is factorized as the sum of the scores of each edge $\psi(w_h, w_m; \theta)$.

Maximum Spanning Tree Decoding. The decoding problem of this parsing model can be formulated as:

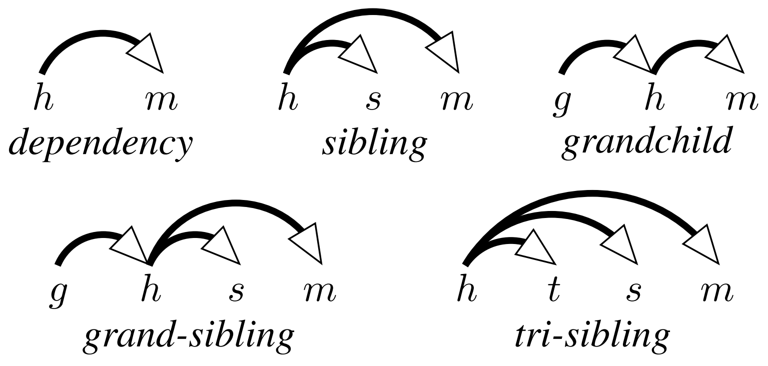
$$\begin{aligned} \mathbf{y}^* &= \underset{\mathbf{y} \in T(\mathbf{x})}{\operatorname{argmax}} \, \psi(\mathbf{y} | \mathbf{x}; \theta) \\ &= \underset{\mathbf{y} \in T(\mathbf{x})}{\operatorname{argmax}} \sum_{(w_h, w_m) \in \mathbf{y}} \psi(w_h, w_m; \theta) \end{aligned}$$

which can be solved by using the Maximum Spanning Tree (MST) algorithm described in McDonald et al. [111].

Higher-order Factorizations. A common strategy to improve the edge-factorization is to utilize high-order factorization:

$$\psi(\mathbf{x}, \mathbf{y}; \theta) = \sum_{p \in \mathbf{y}} \psi(p; \theta)$$

where p is a part of the dependency tree $\mathbf{y}$.



McDonald and Pereira [109] proposed to factorize each tree into second-order *sibling* parts — parts of dependencis consists of a triple of indices (h, m, s) where (h, m) and (h, s) are dependencies, and where s and m are successive modifiers to the same side of h. Koo and Collins

[75] introduced a second-order *grandchild* part — pairs of dependencies connected head-to-tail; and a third-order *grand-sibling* part — combinations of sibling parts and grandchild parts. Further, Ma and Zhao [97] combined grand-sibling and tri-sibling parts to propose the fourth-order *grand-tri-sibling* part.

4.1.2 Transition-based Dependency Parsing

Transition-based dependency parsers, on the other hand, read words sequentially (commonly from left-to-right) and build dependency trees incrementally by making series of multiple choice decisions. A classifier is trained to score the possible decisions at each state of the process and guide the parsing process. The advantage of this formalism is that the number of operations required to build any projective parse tree is linear with respect to the length of the sentence. The challenge, however, is that the decision made at each step is based on local information, leading to error propagation and worse performance compared to graph-based parsers on root and long dependencies [108].

4.2 Neural Probabilistic Model for MST Parsing

4.2.1 Edge-Factored Probabilistic Model

The probabilistic model of NeuroMST parser defines a family of conditional probability $P_\theta(\mathbf{y}|\mathbf{x})$ over all valid parse trees $\mathbf{y}$ given a sentence $\mathbf{x}$, with a log-linear form:

$$P_\theta(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\sum\limits_{(w_h,w_m)\in\mathbf{y}} \psi(w_h, w_m; \theta)\right)}{Z(\mathbf{x}; \theta)}$$

where $Z(\mathbf{x}; \theta)$ is the partition function.

$$Z(\mathbf{x}; \theta) = \sum_{\mathbf{y}\in T(\mathbf{x})} \exp\left(\sum_{(w_h,w_m)\in\mathbf{y}} \psi(w_h, w_m; \theta)\right)$$

Bi-Linear Score Function. In our model, we adopt a bi-linear form score function:

$$\psi(w_h, w_m; \theta) = \phi(w_h)^T \mathbf{W} \phi(w_m) + \mathbf{U}^T \phi(w_h) + \mathbf{V}^T \phi(w_m) + \mathbf{b}$$

where $\{\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}\} \subset \theta$, $\phi(x_i)$ is the representation vector of w_i, $\mathbf{W}, \mathbf{U}, \mathbf{V}$ denote the weight matrix of the bi-linear term and the two weight vectors of the linear terms in ψ, and $\mathbf{b}$ denotes the bias vector.

As discussed in Dozat and Manning [43], the bi-linear form of score function is related to the bi-linear attention mechanism [92]. The bi-linear score function differs from the traditional score function proposed in Kiperwasser and Goldberg [74] by adding the bi-linear term. A similar score function is proposed in Dozat and Manning [43]. The difference between their and our score function is that they only used the linear term for head words ($\mathbf{U}^T\phi(x_h)$) while use them for both heads and modifiers.

Matrix-Tree Theorem. In order to train the probabilistic parsing model, as discussed in Koo et al. [76], we have to compute the *partition function* and the *marginals*, requiring summation over the set $T(\mathbf{x})$:

$$\begin{aligned} Z(\mathbf{x};\theta) &= \sum_{\mathbf{y}\in T(\mathbf{x})} \exp\left(\sum_{(w_h,w_m)\in\mathbf{y}} \psi(w_h, w_m;\theta)\right) \\ \mu_{h,m}(\mathbf{x};\theta) &= \sum_{\mathbf{y}\in T(\mathbf{x}):(w_h,w_m)\in\mathbf{y}} P(\mathbf{y}|\mathbf{x};\theta) \end{aligned}$$

where $\mu_{h,m}(\mathbf{x};\Theta)$ is the marginal for edge from hth word to mth word for $\mathbf{x}$.

Previous studies [76, 137] have presented how a variant of Kirchhoff's Matrix-Tree Theorem [150] can be used to evaluate the partition function and marginals efficiently. In this section, we briefly revisit this method.

For a sentence $\mathbf{x}$ with n words, we denote $\mathbf{x} = \{w_0, w_1, \dots, w_n\}$, where w_0 is the root-symbol. We define a complete graph G on $n+1$ nodes (including the root-symbol x_0), where each node corresponds to a word in $\mathbf{x}$ and each edge corresponds to a dependency arc between two words. Then, we assign non-negative weights to the edges of this complete graph with $n+1$ nodes, yielding the weighted adjacency matrix $\mathbf{A}(\theta) \in \mathbb{R}^{n+1\times n+1}$, for $h, m = 0, \dots, n$:

$$\mathbf{A}_{h,m}(\theta) = \exp\left(\psi(w_h, w_m;\theta)\right)$$

Based on the adjacency matrix $\mathbf{A}(\theta)$, we have the Laplacian matrix:

$$\mathbf{L}(\theta) = \mathbf{D}(\theta) - \mathbf{A}(\theta)$$

where $\mathbf{D}(\theta)$ is the weighted degree matrix:

$$\mathbf{D}_{h,m}(\theta) = \begin{cases} \sum\limits_{h'=0}^{n} \mathbf{A}_{h',m}(\theta) & \text{if } h = m \\ 0 & \text{otherwise} \end{cases}$$

Then, according to Theorem 1 in Koo et al. [76], the partition function is equal to the minor of $\mathbf{L}(\theta)$ w.r.t row 0 and column 0:

$$Z(\mathbf{x};\theta) = \mathbf{L}^{(0,0)}(\theta)$$

where for a matrix $\mathbf{A}$, $\mathbf{A}^{(h,m)}$ denotes the *minor* of $\mathbf{A}$ w.r.t row h and column m; i.e., the determinant of the submatrix formed by deleting the hth row and mth column.

The marginals can be computed by calculating the matrix inversion of the matrix corresponding to $\mathbf{L}^{(0,0)}(\theta)$. The time complexity of computing the partition function and marginals is $O(n^3)$.

Labeled Parsing Model. Though it is originally designed for unlabeled parsing, our probabilistic parsing model is easily extended to include dependency labels.

In labeled dependency trees, each edge is represented by a tuple (w_h, w_m, l), where w_h and w_m are the head word and modifier, respectively, and l is the label of dependency type of this edge. Then we can extend the original model for labeled dependency parsing by extending the score function to include dependency labels:

$$\psi(w_h, w_m, l; \theta) = \phi(w_h)^T \mathbf{W}_l \phi(w_m) + \mathbf{U}_l^T \phi(w_h) + \mathbf{V}_l^T \phi(w_m) + \mathbf{b}_l$$

where $\mathbf{W}_l, \mathbf{U}_l, \mathbf{V}_l, \mathbf{b}_l$ are the weights and bias corresponding to dependency label l. Suppose that there are L different dependency labels, it suffices to define the new adjacency matrix by assigning the weight of a edge with the sum of weights over different dependency labels:

$$\mathbf{A'}_{h,m}(\theta) = \sum_{l=1}^{L} \exp\left(\psi(w_h, w_m, l; \theta)\right)$$

The partition function and marginals over labeled dependency trees are obtained by operating on the new adjacency matrix $\mathbf{A}'(\theta)$. The time complexity becomes $O(n^3 + Ln^2)$. In practice, L is probably large. For English, the number of edge labels in Stanford Basic Dependencies [38] is 45, and the number in the treebank of CoNLL-2008 shared task [143] is 70, while the average length of sentences in English Penn Treebank [104] is around 23. Thus, L is not negligible to n.

It should be noticed that in our labeled model, for different dependency label l we use the same vector representation $\phi(w_i)$ for each word w_i. The dependency labels are distinguished (only) by the parameters (weights and bias) corresponding to each of them. One advantage of this is that it significantly reduces the memory requirement comparing to the model in Dozat and Manning [43] which distinguishes $\phi_l(w_i)$ for different label l.

4.2.2 Neural Representation Encoding

The encoder of our parsing model is based on the bi-directional LSTM-CNN architecture (BLSTM-CNNs) [94] where CNNs encode character-level information of a word into its character-level representation and BLSTM models context information of each word. Formally, for each word, the CNN, with character embeddings as inputs, encodes the character-level representation. Then the character-level representation vector is concatenated with the word embedding vector to feed into the BLSTM network. To enrich word-level information, we also use POS embeddings. Figure 4.1 illustrates the architecture of our network in detail.

4.2.3 Neural Network Training

Parameter Initialization

Word Embeddings. For all the parsing models on different languages, we initialize word vectors with pretrained word embeddings. For Chinese, Dutch, English, German and Spanish, we use the structured-skipgram [88] embeddings, and for other languages we use the Polyglot [3] embeddings. The dimensions of embeddings are 100 for English, 50 for Chinese and 64 for other languages.

Character Embeddings. Following Ma and Hovy [94], character embeddings are initialized with uniform samples from $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$, where we set $dim = 50$.

POS Embedding. Our model also includes POS embeddings. The same as character embeddings, POS embeddings are also 50-dimensional, initialized uniformly from $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$.

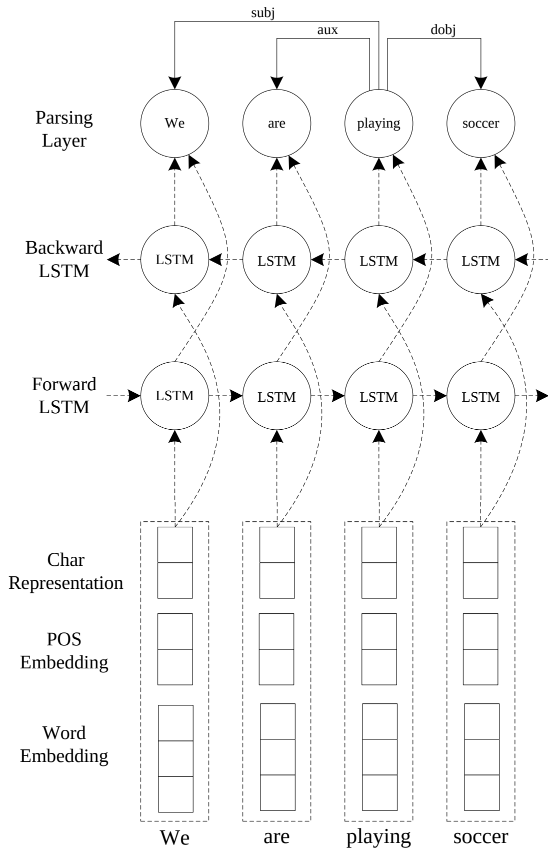


Figure 4.1: The main architecture of our parsing model. The character representation for each word is computed by the CNN in Figure 2.5. Then the character representation vector is concatenated with the word and pos embedding before feeding into the BLSTM network. Dashed arrows indicate dropout layers applied on the input, hidden and output vectors of BLSTM.

Weights Matrices and Bias Vectors. Matrix parameters are randomly initialized with uniform samples from $[-\sqrt{\frac{6}{r+c}}, +\sqrt{\frac{6}{r+c}}]$, where r and c are the number of of rows and columns in the structure [57]. Bias vectors are initialized to zero, except the bias $\mathbf{b}_f$ for the forget gate in LSTM , which is initialized to 1.0 [70].

Optimization Algorithm

Parameter optimization is performed with the Adam optimizer [73] with $\beta1 = \beta2 = 0.9$. We choose an initial learning rate of $\eta_0 = 0.002$. The learning rate η was adapted using a schedule $S = [e_1, e_2, \dots, e_s]$, in which the learning rate η is annealed by multiplying a fixed decay rate

	English				Chinese				German			
	Dev		**Test**		**Dev**		**Test**		**Dev**		**Test**	
Model	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Basic	94.51	92.23	94.62	92.54	84.33	81.65	84.35	81.63	90.46	87.77	90.69	88.42
+Char	94.74	92.55	94.73	92.75	85.07	82.63	85.24	82.46	92.16	89.82	92.24	90.18
+POS	94.71	92.60	94.83	92.96	88.98	87.55	89.05	87.74	91.94	89.51	92.19	90.05
Full	94.77	92.66	94.88	92.98	88.51	87.16	88.79	87.47	92.37	90.09	92.58	90.54

Table 4.1: Parsing performance (UAS and LAS) of different versions of our model on both the development and test sets for three languages.

$\rho = 0.5$ after $e_i \in S$ epochs respectively. We used $S = [10, 30, 50, 70, 100]$ and trained all networks for a total of 120 epochs. While the Adam optimizer automatically adjusts the global learning rate according to past gradient magnitudes, we find that this additional decay consistently improves model performance across all settings and languages. To reduce the effects of "gradient exploding", we use a gradient clipping of 5.0 [123]. We explored other optimization algorithms such as stochastic gradient descent (SGD) with momentum, AdaDelta [159], or RMSProp [37], but none of them meaningfully improve upon Adam with learning rate annealing in our preliminary experiments. Meanwhile, Adam significantly accelerates the training procedure.

Dropout Training. To mitigate overfitting, we apply the dropout method [100, 140] to regularize our model. As shown in 4.1, we apply dropout on character embeddings before inputting to CNN, and on the input, hidden and output vectors of BLSTM. We apply dropout rate of 0.15 to all the embeddings. For BLSTM, we use the recurrent dropout [52] with 0.25 dropout rate between hidden states and 0.33 between layers. We found that the model using the new recurrent dropout converged much faster than standard dropout, while achiving similar performance.

4.2.4 Experiments Setup

We evaluate our neural probabilistic parser on the same data setup as Kuncoro et al. [79], namely the English Penn Treebank (PTB version 3.0) [104], the Penn Chinese Treebank (CTB version 5.1) [157], and the German CoNLL 2009 corpus [63]. Following previous work, all experiments are evaluated on the metrics of unlabeled and labeled attachment score (UAS and LAS).

4.2.5 Main Results

We first construct experiments to dissect the effectiveness of each input information (embeddings) of our neural network architecture by ablation studies. We compare the performance of four versions of our model with different inputs — Basic, +POS, +Char and Full — where the Basic model utilizes only the pretrained word embeddings as inputs, while the +POS and +Char models augments the basic one with POS embedding and character information, respectively. According to the results shown in Table 4.1, +Char model obtains better performance than the Basic model on all the three languages, showing that character-level representations are important for dependency parsing. Second, on English and German, +Char and +POS achieves comparable performance, while on Chinese +POS significantly outperforms +Char model. Finally, the Full model achieves the best accuracy on English and German, but on Chinese +POS obtains the best. Thus, we guess that the POS information is more useful for Chinese than English and German.

	Dev		**Test**	
	UAS	LAS	UAS	LAS
cross-entropy	94.10	91.52	93.77	91.57
global-likelihood	94.77	92.66	94.88	92.98

Table 4.2: Parsing performance on PTB with different training objective functions.

Table 4.2 gives the performance on PTB of the parsers trained with two different objective functions — the cross-entropy objective of each word, and our objective based on likelihood for an entire tree. The parser with global likelihood objective outperforms the one with simple cross-entropy objective, demonstrating the effectiveness of the global structured objective.

Comparison with Previous Work

Table 4.3 illustrates the results of the four versions of our model on the three languages, together with twelve previous top-performance systems for comparison. Our Full model significantly

	English		Chinese		German	
System	UAS	LAS	UAS	LAS	UAS	LAS
Bohnet and Nivre [20]	–	–	87.3	85.9	91.4	89.4
Chen and Manning [23]	91.8	89.6	83.9	82.4	–	–
Ballesteros et al. [10]	91.6	89.4	85.3	83.7	88.8	86.1
Dyer et al. [45]	93.1	90.9	87.2	85.7	–	–
Kiperwasser and Goldberg [74]: graph	93.1	91.0	86.6	85.1	–	–
Ballesteros et al. [11]	93.6	91.4	87.7	86.2	–	–
Wang and Chang [152]	94.1	91.8	87.6	86.2	–	–
Zhang et al. [162]	94.1	91.9	87.8	86.2	–	–
Cheng et al. [24]	94.1	91.5	88.1	85.7	–	–
Andor et al. [6]	94.6	92.8	–	–	90.9	89.2
Kuncoro et al. [79]	94.3	92.1	88.9	87.3	91.6	89.2
Dozat and Manning [43]	**95.7**	**94.1**	**89.3**	**88.2**	**93.5**	**91.4**
This work: Basic	94.6	92.5	84.4	81.6	90.7	88.4
This work: +Char	94.7	92.8	85.2	82.5	92.2	90.2
This work: +POS	94.8	93.0	89.1	87.7	92.2	90.1
This work: Full	94.9	93.0	88.8	87.5	92.6	90.5

Table 4.3: UAS and LAS of four versions of our model on test sets for three languages, together with top-performance parsing systems.

outperforms the graph-based parser proposed in Kiperwasser and Goldberg [74] which used similar neural network architecture for representation learning. Moreover, our model achieves better results than the parser distillation method [79] on all the three languages. The results of our parser are slightly worse than the scores reported in Dozat and Manning [43]. One possible reason is that for labeled dependency parsing Dozat and Manning [43] used different vectors for different dependency labels to represent each word, making their model require much more memory.

	Turbo	**Tensor**	**RGB**	**In-Out**	**Bi-Att**	**+POS**	**Full**	**Best**
	UAS	UAS	UAS	UAS [LAS]	UAS [LAS]	UAS [LAS]	UAS [LAS]	UAS
ar	79.64	79.95	80.24	79.60 [67.09]	80.34 [68.58]	80.05 [67.80]	80.80 [**69.40**]	**81.12**
bg	93.10	93.50	93.72	92.68 [87.79]	93.96 [89.55]	93.66 [89.79]	**94.28** [**90.60**]	94.02
zh	89.98	92.68	93.04	92.58 [88.51]	–	**93.44** [90.04]	93.40 [**90.10**]	93.04
cs	90.32	90.50	90.77	88.01 [79.31]	91.16 [85.14]	91.04 [85.82]	**91.18** [**85.92**]	91.16
da	91.48	91.39	91.86	91.44 [85.55]	91.56 [85.53]	91.52 [86.57]	91.86 [**87.07**]	**92.00**
nl	86.19	86.41	87.39	84.45 [80.31]	87.15 [82.41]	87.41 [84.17]	**87.85** [**84.82**]	87.39
en	93.22	93.02	93.25	92.45 [89.43]	–	94.43 [92.31]	**94.66** [**92.52**]	93.25
de	92.41	91.97	92.67	90.79 [87.74]	92.71 [89.80]	93.53 [91.55]	**93.62** [**91.90**]	92.71
ja	93.52	93.71	93.56	93.54 [91.80]	93.44 [90.67]	93.82 [92.34]	**94.02** [**92.60**]	93.80
pt	92.69	91.92	92.36	91.54 [87.68]	92.77 [88.44]	92.59 [**89.12**]	92.71 [88.92]	**93.03**
sl	86.01	86.24	86.72	84.39 [73.74]	86.01 [75.90]	85.73 [76.48]	86.73 [**77.56**]	**87.06**
es	85.59	88.00	88.75	86.44 [83.29]	88.74 [84.03]	88.58 [85.03]	**89.20** [**85.77**]	88.75
sv	91.14	91.00	91.08	89.94 [83.09]	90.50 [84.05]	90.89 [86.58]	91.22 [**86.92**]	**91.85**
tr	76.90	76.84	76.68	75.32 [60.39]	**78.43** [**66.16**]	75.88 [61.72]	77.71 [65.81]	78.43
av	88.73	89.08	89.44	88.08 [81.84]	–	89.47 [84.24]	89.95 [84.99]	89.83

Table 4.4: UAS and LAS on 14 treebanks from CoNLL shared tasks, together with several state-of-the-art parsers. "Best Published" includes the most accurate parsers in term of UAS among Koo et al. [77], Martins et al. [105], Martins et al. [106], Lei et al. [85], Zhang et al. [163], Zhang and McDonald [160], Pitler and McDonald [131], Ma and Hovy [93], and Cheng et al. [24].

Experiments on CoNLL Treebanks

Datasets. To make a thorough empirical comparison with previous studies, we also evaluate our system on treebanks from CoNLL shared task on dependency parsing — the English treebank from CoNLL-2008 shared task [143] and all 13 treebanks from CoNLL-2006 shared task [21]. For the treebanks from CoNLL-2006 shared task, following Cheng et al. [24], we randomly select 5% of the training data as the development set. UAS and LAS are evaluated using the official scorer[1] of CoNLL-2006 shared task.

[1]`http://ilk.uvt.nl/conll/software.html`

Baselines. We compare our model with the third-order Turbo parser [106], the low-rank tensor based model (Tensor) [85], the randomized greedy inference based (RGB) model [163], the labeled dependency parser with inner-to-outer greedy decoding algorithm (In-Out) [93], and the bi-direction attention based parser (Bi-Att) [24]. We also compare our parser against the best published results for individual languages. This comparison includes four additional systems: Koo et al. [77], Martins et al. [105], Zhang and McDonald [160] and Pitler and McDonald [131].

Results. Table 4.4 summarizes the results of our model, along with the state-of-the-art baselines. On average across 14 languages, our approach significantly outperforms all the baseline systems. It should be noted that the average UAS of our parser over the 14 languages is better than that of the "best published", which are from different systems that achieved best results for different languages.

For individual languages, our parser achieves state-of-the-art performance on both UAS and LAS on 8 languages — Bulgarian, Chinese, Czech, Dutch, English, German, Japanese and Spanish. On Arabic, Danish, Portuguese, Slovene and Swedish, our parser obtains the best LAS. Another interesting observation is that the Full model outperforms the +POS model on 13 languages. The only exception is Chinese, which matches the observation in Section 4.2.5.

4.3 Discussions

In this chapter, we proposed a neural probabilistic model for non-projective dependency parsing, which combined the BLSTM-CNNs architecture for representation learning with a probabilistic structured output layer on top. Experimental results on 17 treebanks across 14 languages show that our parser significantly improves the accuracy of both dependency structures (UAS) and edge labels (LAS), over several previously state-of-the-art systems.

In the next chapter, we will consider neural networks for transition-based dependency parsing, where we introduce our stack-pointer network.

Chapter 5

Stack-Pointer Networks for Dependency Parsing

In the last chaper, we showd that incorporating this global search algorithm with distributed representations learned from neural networks, neural graph-based parsers [43, 74, 79, 152] have achieved the state-of-the-art accuracies on a number of treebanks in different languages. Nevertheless, these models, while accurate, are usually slow (e.g. decoding is $O(n^3)$ time complexity for first-order models [110, 111] and higher polynomials for higher-order models [75, 97, 98, 109]).

Transition-based dependency parsers, on the other hand, read words sequentially (commonly from left-to-right) and build dependency trees incrementally by making series of multiple choice decisions. A classifier is trained to score the possible decisions at each state of the process and guide the parsing process. The advantage of this formalism is that the number of operations required to build any projective parse tree is linear with respect to the length of the sentence. The challenge, however, is that the decision made at each step is based on local information, leading to error propagation and worse performance compared to graph-based parsers on root and long dependencies [108]. Previous studies have explored solutions to address this challenge. Stack LSTMs [10, 11, 45] are capable of learning representations of the parser state that are sensitive to the complete contents of the parser's state. Andor et al. [6] proposed a globally normalized transition model to replace the locally normalized classifier. However, the parsing accuracy is still behind state-of-the-art graph-based parsers [43].

In this chapter, we propose a novel neural network architecture for dependency parsing, *stack-pointer networks* (**STACKPTR**). STACKPTR is a transition-based architecture, with the corresponding asymptotic efficiency, but still maintains a global view of the sentence that proves essential for achieving competitive accuracy. Our STACKPTR parser has a pointer network [151] as its backbone, and is equipped with an internal stack to maintain the order of head words in tree structures. The STACKPTR parser performs parsing in an incremental, top-down, depth-first fashion; at each step, it generates an arc by assigning a child for the head word at the top of the internal stack. This architecture makes it possible to capture information from the whole sentence and all the previously derived subtrees, while maintaining a number of parsing steps linear in the sentence length.

We evaluate our parser on 29 treebanks across 20 languages and different dependency annotation schemas, and achieve state-of-the-art performance on 21 of them. The contributions of this work are summarized as follows:

(i) We propose a neural network architecture for dependency parsing that is simple, effective, and efficient.

(ii) Empirical evaluations on benchmark datasets over 20 languages show that our method achieves state-of-the-art performance on 21 different treebanks[1].

(iii) Comprehensive error analysis is conducted to compare the proposed method to a strong graph-based baseline using biaffine attention [43].

5.1 Background

5.1.1 Notations

Dependency trees represent syntactic relationships between words in the sentences through labeled directed edges between head words and their dependents. Figure 5.1 (a) shows a dependency tree for the sentence, "But there were no buyers".

[1]Source code is publicly available at `https://github.com/XuezheMax/NeuroNLP2`

In this paper, we will use the following notation:

Input: $\mathbf{x} = \{w_1, \ldots, w_n\}$ represents a generic sentence, where w_i is the ith word.

Output: $\mathbf{y} = \{p_1, p_2, \cdots, p_k\}$ represents a generic (possibly non-projective) dependency tree, where each path $p_i = \$, w_{i,1}, w_{i,2}, \cdots, w_{i,l_i}$ is a sequence of words from the root to a leaf. "\$" is an universal virtual root that is added to each tree.

Stack: σ denotes a stack configuration, which is a sequence of words. We use $\sigma|w$ to represent a stack configuration that pushes word w into the stack σ.

Children: $\mathrm{ch}(w_i)$ denotes the list of all the children (modifiers) of word w_i.

5.1.2 Pointer Networks

Pointer Networks (PTR-NET) [151] are a variety of neural network capable of learning the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. This model cannot be trivially expressed by standard sequence-to-sequence networks [144] due to the variable number of input positions in each sentence. PTR-NET solves the problem by using attention [9, 92] as a pointer to select a member of the input sequence as the output.

Formally, the words of the sentence $\mathbf{x}$ are fed one-by-one into the encoder (a multiple-layer bi-directional RNN), producing a sequence of *encoder hidden states* s_i. At each time step t, the decoder (a uni-directional RNN) receives the input from last step and outputs *decoder hidden state* h_t. The *attention vector* a^t is calculated as follows:

$$\begin{aligned} e_i^t &= score(h_t, s_i) \\ a^t &= softmax(e^t) \end{aligned} \tag{5.1}$$

where $score(\cdot, \cdot)$ is the *attention scoring function*, which has several variations such as dot-product, concatenation, and biaffine [92]. PTR-NET regards the attention vector a^t as a probability distribution over the source words, i.e. it uses a_i^t as pointers to select the input elements.

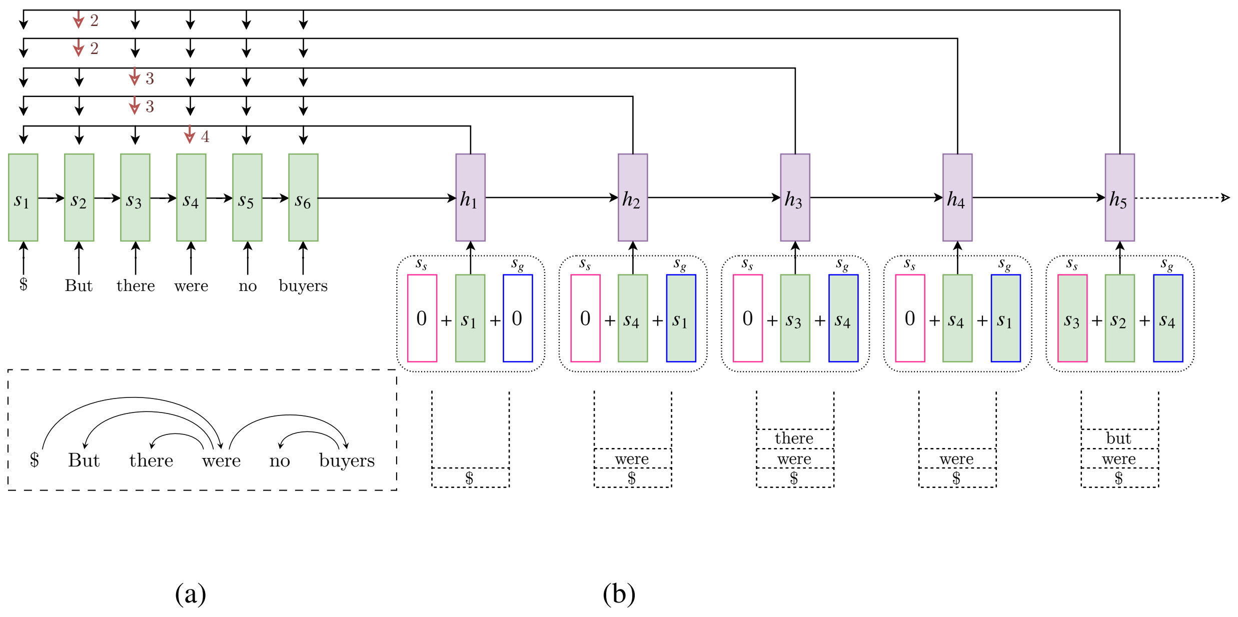


Figure 5.1: Neural architecture for the StackPtr network, together with the decoding procedure of an example sentence. The BiRNN of the encoder is elided for brevity. For the inputs of decoder at each time step, vectors in red and blue boxes indicate the sibling and grandparent.

5.2 Stack-Pointer Networks

5.2.1 Overview

Similarly to Ptr-Net, StackPtr first reads the whole sentence and encodes each word into the encoder hidden state s_i. The internal stack σ is always initialized with the root symbol \$. At each time step t, the decoder receives the input vector corresponding to the top element of the stack σ (the head word w_p where p is the word index), generates the hidden state h_t, and computes the attention vector a^t using Eq. (5.1). The parser chooses a specific position c according to the attention scores in a^t to generate a new dependency arc (w_h, w_c) by selecting w_c as a child of w_h. Then the parser pushes w_c onto the stack, i.e. $\sigma \rightarrow \sigma|w_c$, and goes to the next step. At one step if the parser points w_h to itself, i.e. $c = h$, it indicates that all children of the head word w_h have already been selected. Then the parser goes to the next step by popping w_h out of σ.

At test time, in order to guarantee a valid dependency tree containing all the words in the input sentences exactly once, the decoder maintains a list of "available" words. At each decoding step, the parser selects a child for the current head word, and removes the child from the list of available words to make sure that it cannot be selected as a child of other head words.

For head words with multiple children, it is possible that there is more than one valid selection for each time step. In order to define a deterministic decoding process to make sure that there is only one ground-truth choice at each step (which is necessary for simple maximum likelihood estimation), a predefined order for each $\mathrm{ch}(w_i)$ needs to be introduced. The predefined order of children can have different alternatives, such as left-to-right or inside-out[2]. In this paper, we adopt the inside-out order[3] since it enables us to utilize second-order *sibling* information, which has been proven beneficial for parsing performance [75, 109] (see § 5.2.4 for details). Figure 5.1 (b) depicts the architecture of STACKPTR and the decoding procedure for the example sentence in Figure 5.1 (a).

5.2.2 Encoder

The encoder of our parsing model is based on the bi-directional LSTM-CNN architecture (BLSTM-CNNs) [94] where CNNs encode character-level information of a word into its character-level representation and BLSTM models context information of each word. Formally, for each word, the CNN, with character embeddings as inputs, encodes the character-level representation. Then the character-level representation vector is concatenated with the word embedding vector to feed into the BLSTM network. To enrich word-level information, we also use POS embeddings. Finally, the encoder outputs a sequence of hidden states s_i.

5.2.3 Decoder

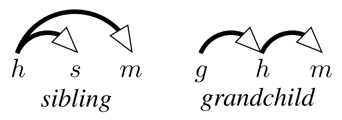
The decoder for our parser is a uni-directional LSTM. Different from previous work [9, 151] which uses word embeddings of the previous word as the input to the decoder, our decoder receives the encoder hidden state vector (s_i) of the top element in the stack σ (see Figure 5.1 (b)). Compared to word embeddings, the encoder hidden states contain more contextual information, benefiting

[2]Order the children by the distances to the head word on the left side, then the right side.
[3]We also tried left-to-right order which obtained worse parsing accuracy than inside-out.

both the training and decoding procedures. The decoder produces a sequence of decoder hidden states h_i, one for each decoding step.

5.2.4 Higher-order Information

As mentioned before, our parser is capable of utilizing higher-order information. In this paper, we incorporate two kinds of higher-order structures — *grandparent* and *sibling*. A sibling structure is a head word with two successive modifiers, and a grandparent structure is a pair of dependencies connected head-to-tail:



To utilize higher-order information, the decoder's input at each step is the sum of the encoder hidden states of three words:

$$\beta_t = s_h + s_g + s_s$$

where β_t is the input vector of decoder at time t and h, g, s are the indices of the head word and its grandparent and sibling, respectively. Figure 5.1 (b) illustrates the details. Here we use the element-wise sum operation instead of concatenation because it does not increase the dimension of the input vector β_t, thus introducing no additional model parameters.

5.2.5 Biaffine Attention Mechanism

For attention score function (Eq. (5.1)), we adopt the biaffine attention mechanism [43, 92]:

$$e_i^t = h_t^T \mathbf{W} s_i + \mathbf{U}^T h_t + \mathbf{V}^T s_i + \mathbf{b}$$

where $\mathbf{W}, \mathbf{U}, \mathbf{V}$, $\mathbf{b}$ are parameters, denoting the weight matrix of the bi-linear term, the two weight vectors of the linear terms, and the bias vector.

As discussed in Dozat and Manning [43], applying a multilayer perceptron (MLP) to the output vectors of the BLSTM before the score function can both reduce the dimensionality and overfitting of the model. We follow this work by using a one-layer perceptron to s_i and h_i with ELU [28] as its activation function.

Similarly, the dependency label classifier also uses a biaffine function to score each label, given the head word vector h_t and child vector s_i as inputs. Again, we use MLPs to transform h_t and s_i before feeding them into the classifier.

5.2.6 Training Objectives

The STACKPTR parser is trained to optimize the probability of the dependency trees given sentences: $P_\theta(\mathbf{y}|\mathbf{x})$, which can be factorized as:

$$P_\theta(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{k} P_\theta(p_i|p_{<i}, \mathbf{x}) = \prod_{i=1}^{k} \prod_{j=1}^{l_i} P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, \mathbf{x}), \tag{5.2}$$

where θ represents model parameters. $p_{<i}$ denotes the preceding paths that have already been generated. $c_{i,j}$ represents the jth word in p_i and $c_{i,<j}$ denotes all the proceeding words on the path p_i. Thus, the STACKPTR parser is an autoregressive model, like sequence-to-sequence models, but it factors the distribution according to a top-down tree structure as opposed to a left-to-right chain. We define $P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, \mathbf{x}) = a^t$, where attention vector a^t (of dimension n) is used as the distribution over the indices of words in a sentence.

Arc Prediction Our parser is trained by optimizing the conditional likelihood in Eq (5.2), which is implemented as the cross-entropy loss.

Label Prediction We train a separated multi-class classifier in parallel to predict the dependency labels. Following Dozat and Manning [43], the classifier takes the information of the head word and its child as features. The label classifier is trained simultaneously with the parser by optimizing the sum of their objectives.

5.2.7 Discussion

Time Complexity. The number of decoding steps to build a parse tree for a sentence of length n is $2n-1$, linear in n. Together with the attention mechanism (at each step, we need to compute the attention vector a^t, whose runtime is $O(n)$), the time complexity of decoding algorithm is

$O(n^2)$, which is more efficient than graph-based parsers that have $O(n^3)$ or worse complexity when using dynamic programming or maximum spanning tree (MST) decoding algorithms.

Top-down Parsing. When humans comprehend a natural language sentence, they arguably do it in an incremental, left-to-right manner. However, when humans consciously annotate a sentence with syntactic structure, they rarely ever process in fixed left-to-right order. Rather, they start by reading the whole sentence, then seeking the main predicates, jumping back-and-forth over the sentence and recursively proceeding to the sub-tree structures governed by certain head words. Our parser follows a similar kind of annotation process: starting from reading the whole sentence, and processing in a top-down manner by finding the main predicates first and only then search for sub-trees governed by them. When making latter decisions, the parser has access to the entire structure built in earlier steps.

5.2.8 Implementation Details

Pre-trained Word Embeddings. For all the parsing models in different languages, we initialize word vectors with pretrained word embeddings. For Chinese, Dutch, English, German and Spanish, we use the structured-skipgram [88] embeddings. For other languages we use Polyglot embeddings [3].

Optimization. Parameter optimization is performed with the Adam optimizer [73] with $\beta_1 = \beta_2 = 0.9$. We choose an initial learning rate of $\eta_0 = 0.001$. The learning rate η is annealed by multiplying a fixed decay rate $\rho = 0.75$ when parsing performance stops increasing on validation sets. To reduce the effects of "gradient exploding", we use gradient clipping of 5.0 [123].

Dropout Training. To mitigate overfitting, we apply dropout [100, 140]. For BLSTM, we use recurrent dropout [52] with a drop rate of 0.33 between hidden states and 0.33 between layers. Following Dozat and Manning [43], we also use embedding dropout with a rate of 0.33 on all word, character, and POS embeddings.

5.3 Experiments

5.3.1 Setup

We evaluate our STACKPTR parser mainly on three treebanks: the English Penn Treebank (PTB version 3.0) [104], the Penn Chinese Treebank (CTB version 5.1) [157], and the German CoNLL 2009 corpus [63]. We use the same experimental settings as Kuncoro et al. [79].

To make a thorough empirical comparison with previous studies, we also evaluate our system on treebanks from CoNLL shared task and the Universal Dependency (UD) Treebanks[4]. For the CoNLL Treebanks, we use the English treebank from CoNLL-2008 shared task [143] and all 13 treebanks from CoNLL-2006 shared task [21]. The experimental settings are the same as Ma and Hovy [93]. For UD Treebanks, we select 12 languages. The details of the treebanks and experimental settings are in § 5.3.6 and Appendix A.1.2.

Evaluation Metrics Parsing performance is measured with five metrics: unlabeled attachment score (UAS), labeled attachment score (LAS), unlabeled complete match (UCM), labeled complete match (LCM), and root accuracy (RA). Following previous work [43, 79], we report results excluding punctuations for Chinese and English. For each experiment, we report the mean values with corresponding standard deviations over 5 repetitions.

Baseline For fair comparison of the parsing performance, we re-implemented the graph-based Deep Biaffine (BIAF) parser [43], which achieved state-of-the-art results on a wide range of languages. Our re-implementation adds character-level information using the same LSTM-CNN encoder as our model (§ 5.2.2) to the original BIAF model, which boosts its performance on all languages.

5.3.2 Main Results

We first conduct experiments to demonstrate the effectiveness of our neural architecture by comparing with the strong baseline BIAF. We compare the performance of four variations of

[4]http://universaldependencies.org/

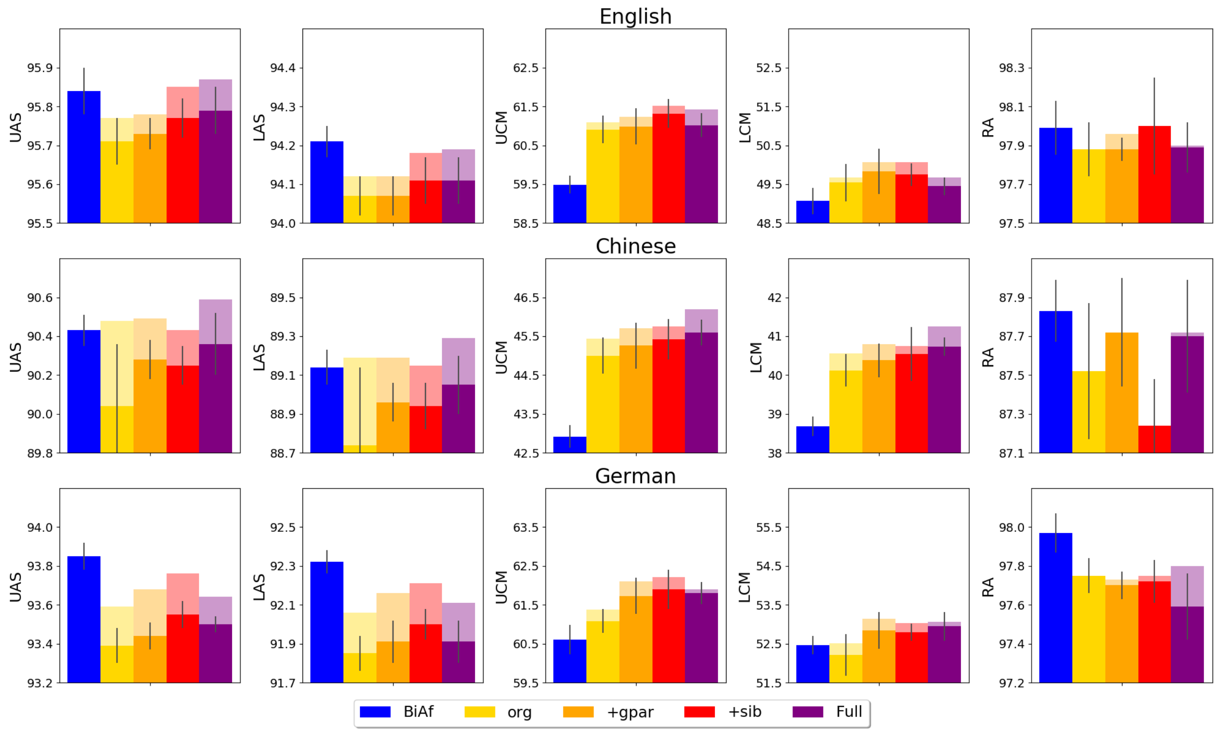


Figure 5.2: Parsing performance of different variations of our model on the test sets for three languages, together with baseline BIAF. For each of our STACKPTR models, we perform decoding with beam size equal to 1 and 10. The improvements of decoding with beam size 10 over 1 are presented by stacked bars with light colors.

our model with different decoder inputs — Org, +gpar, +sib and Full — where the Org model utilizes only the encoder hidden states of head words, while the +gpar and +sib models augments the original one with grandparent and sibling information, respectively. The Full model includes all the three information as inputs.

Figure 5.2 illustrates the performance (five metrics) of different variations of our STACKPTR parser together with the results of baseline BIAF re-implemented by us, on the test sets of the three languages. On UAS and LAS, the Full variation of STACKPTR with decoding beam size 10 outperforms BIAF on Chinese, and obtains competitive performance on English and German. An interesting observation is that the Full model achieves the best accuracy on English and Chinese, while performs slightly worse than +sib on German. This shows that the importance of higher-order information varies in languages. On LCM and UCM, STACKPTR significantly

System		English		Chinese		German	
		UAS	LAS	UAS	LAS	UAS	LAS
Chen and Manning [23]	T	91.8	89.6	83.9	82.4	–	–
Ballesteros et al. [10]	T	91.63	89.44	85.30	83.72	88.83	86.10
Dyer et al. [45]	T	93.1	90.9	87.2	85.7	–	–
Bohnet and Nivre [20]	T	93.33	91.22	87.3	85.9	91.4	89.4
Ballesteros et al. [11]	T	93.56	91.42	87.65	86.21	–	–
Kiperwasser and Goldberg [74]	T	93.9	91.9	87.6	86.1	–	–
Weiss et al. [154]	T	94.26	92.41	–	–	–	–
Andor et al. [6]	T	94.61	92.79	–	–	90.91	89.15
Kiperwasser and Goldberg [74]	G	93.1	91.0	86.6	85.1	–	–
Wang and Chang [152]	G	94.08	91.82	87.55	86.23	–	–
Cheng et al. [24]	G	94.10	91.49	88.1	85.7	–	–
Kuncoro et al. [79]	G	94.26	92.06	88.87	87.30	91.60	89.24
Ma and Hovy [95]	G	94.88	92.98	89.05	87.74	92.58	90.54
BIAF: Dozat and Manning [43]	G	95.74	94.08	89.30	88.23	93.46	91.44
BIAF: re-impl	G	95.84	**94.21**	90.43	89.14	**93.85**	**92.32**
STACKPTR: Org	T	95.77	94.12	90.48	89.19	93.59	92.06
STACKPTR: +gpar	T	95.78	94.12	90.49	89.19	93.65	92.12
STACKPTR: +sib	T	95.85	94.18	90.43	89.15	93.76	92.21
STACKPTR: Full	T	**95.87**	94.19	**90.59**	**89.29**	93.65	92.11

Table 5.1: UAS and LAS of four versions of our model on test sets for three languages, together with top-performing parsing systems. “T” and “G” indicate transition- and graph-based models, respectively. For BIAF, we provide the original results reported in Dozat and Manning [43] and our re-implementation. For STACKPTR and our re-implementation of BiAF, we report the average over 5 runs.

outperforms BIAF on all languages, showing the superiority of our parser on complete sentence parsing. The results of our parser on RA are slightly worse than BIAF. More details of results are provided in Appendix A.1.2.

5.3.3 Comparison with Previous Work

Table 5.1 illustrates the UAS and LAS of the four versions of our model (with decoding beam size 10) on the three treebanks, together with previous top-performing systems for comparison. Note that the results of STACKPTR and our re-implementation of BIAF are the average of 5 repetitions instead of a single run. Our Full model significantly outperforms all the transition-based parsers on all three languages, and achieves better results than most graph-based parsers. Our re-implementation of BIAF obtains better performance than the original one in Dozat and Manning [43], demonstrating the effectiveness of the character-level information. Our model achieves state-of-the-art performance on both UAS and LAS on Chinese, and best UAS on English. On German, the performance is competitive with BIAF, and significantly better than other models.

5.3.4 Error Analysis

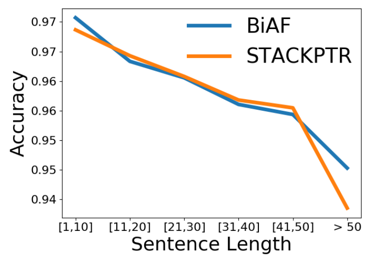
In this section, we characterize the errors made by BIAF and STACKPTR by presenting a number of experiments that relate parsing errors to a set of linguistic and structural properties. For simplicity, we follow McDonald and Nivre [108] and report labeled parsing metrics (either accuracy, precision, or recall) for all experiments.

Length and Graph Factors

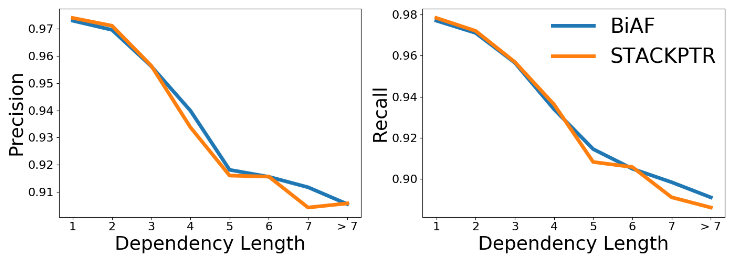
Following McDonald and Nivre [108], we analyze parsing errors related to structural factors.

Sentence Length. Figure 5.3 (a) shows the accuracy of both parsing models relative to sentence lengths. Consistent with the analysis in McDonald and Nivre [108], STACKPTR tends to perform better on shorter sentences, which make fewer parsing decisions, significantly reducing the chance of error propagation.

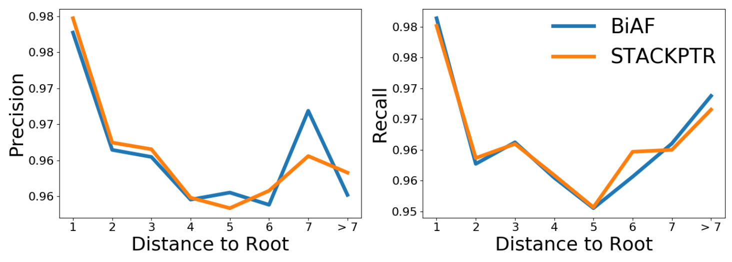
Dependency Length. Figure 5.3 (b) measures the precision and recall relative to dependency lengths. While the graph-based BIAF parser still performs better for longer dependency arcs and transition-based STACKPTR parser does better for shorter ones, the gap between the two systems is marginal, much smaller than that shown in McDonald and Nivre [108]. One possible reason is



(a)



(b)



(c)

Figure 5.3: Performance of BIAF and STACKPTR parsers relative to length and graph factors.

that, unlike traditional transition-based parsers that scan the sentence from left to right, STACKPTR processes in a top-down manner, thus sometimes unnecessarily creating shorter dependency arcs first.

Root Distance. Figure 5.3 (c) plots the precision and recall of each system for arcs of varying distance to the root. Different from the observation in McDonald and Nivre [108], STACKPTR does not show an obvious advantage on the precision for arcs further away from the root. Furthermore, the STACKPTR parser does not have the tendency to over-predict root modifiers reported in McDonald and Nivre [108]. This behavior can be explained using the same reasoning as above: the fact that arcs further away from the root are usually constructed early in the parsing algorithm

POS	UAS	LAS	UCM	LCM
Gold	96.12±0.03	95.06±0.05	62.22±0.33	55.74±0.44
Pred	95.87±0.04	94.19±0.04	61.43±0.49	49.68±0.47
None	95.90±0.05	94.21±0.04	61.58±0.39	49.87±0.46

Table 5.2: Parsing performance on the test data of PTB with different versions of POS tags.

of traditional transition-based parsers is not true for the STACKPTR parser.

Effect of POS Embedding

The only prerequisite information that our parsing model relies on is POS tags. With the goal of achieving an end-to-end parser, we explore the effect of POS tags on parsing performance. We run experiments on PTB using our STACKPTR parser with gold-standard and predicted POS tags, and without tags, respectively. STACKPTR in these experiments is the Full model with beam=10.

Table 5.2 gives results of the parsers with different versions of POS tags on the test data of PTB. The parser with gold-standard POS tags significantly outperforms the other two parsers, showing that dependency parsers can still benefit from accurate POS information. The parser with predicted (imperfect) POS tags, however, performs even slightly worse than the parser without using POS tags. It illustrates that an end-to-end parser that doesn't rely on POS information can obtain competitive (or even better) performance than parsers using imperfect predicted POS tags, even if the POS tagger is relative high accuracy (accuracy $> 97\%$ in this experiment on PTB).

5.3.5 Experiments on CoNLL Treebanks

Table 5.3 summarizes the parsing results of our model on the test sets of 14 treebanks from the CoNLL shared task, along with the state-of-the-art baselines. Along with BIAF, we also list the performance of the bi-directional attention based Parser (Bi-Att) [24] and the neural MST parser (NeuroMST) [95] for comparison. Our parser achieves state-of-the-art performance on both UAS and LAS on eight languages — Arabic, Czech, English, German, Portuguese, Slovene, Spanish, and Swedish. On Bulgarian and Dutch, our parser obtains the best UAS. On other languages, the

	Bi-Att	NeuroMST	BIAF	STACKPTR	**Best Published**	
	UAS [LAS]	UAS [LAS]	UAS [LAS]	UAS [LAS]	UAS	LAS
ar	80.34 [68.58]	80.80 [69.40]	82.15±0.34 [71.32±0.36]	**83.04±0.29 [72.94±0.31]**	81.12	–
bg	93.96 [89.55]	94.28 [90.60]	94.62±0.14 [**91.56±0.24**]	**94.66±0.10** [91.40±0.08]	94.02	–
zh	–	93.40 [90.10]	**94.05±0.27 [90.89±0.22]**	93.88±0.24 [90.81±0.55]	93.04	–
cs	91.16 [85.14]	91.18 [85.92]	92.24±0.22 [87.85±0.21]	**92.83±0.13 [88.75±0.16]**	91.16	85.14
da	91.56 [85.53]	91.86 [87.07]	**92.80±0.26 [88.36±0.18]**	92.08±0.15 [87.29±0.21]	92.00	–
nl	87.15 [82.41]	87.85 [84.82]	90.07±0.18 [**87.24±0.17**]	**90.10±0.27** [87.05±0.26]	87.39	–
en	–	94.66 [92.52]	95.19±0.05 [93.14±0.05]	**93.25±0.05 [93.17±0.05]**	93.25	–
de	92.71 [89.80]	93.62 [91.90]	94.52±0.11 [93.06±0.11]	**94.77±0.05 [93.21±0.10]**	92.71	89.80
ja	93.44 [90.67]	**94.02 [92.60]**	93.95±0.06 [92.46±0.07]	93.38±0.08 [91.92±0.16]	93.80	–
pt	92.77 [88.44]	92.71 [88.92]	93.41±0.08 [89.96±0.24]	**93.57±0.12 [90.07±0.20]**	93.03	–
sl	86.01 [75.90]	86.73 [77.56]	87.55±0.17 [78.52±0.35]	**87.59±0.36 [78.85±0.53]**	87.06	–
es	88.74 [84.03]	89.20 [85.77]	90.43±0.13 [87.08±0.14]	**90.87±0.26 [87.80±0.31]**	88.75	84.03
sv	90.50 [84.05]	91.22 [86.92]	92.22±0.15 [88.44±0.17]	**92.49±0.21 [89.01±0.22]**	91.85	85.26
tr	78.43 [66.16]	77.71 [65.81]	**79.84±0.23 [68.63±0.29]**	79.56±0.22 [68.03±0.15]	78.43	66.16

Table 5.3: UAS and LAS on 14 treebanks from CoNLL shared tasks, together with several state-of-the-art parsers. Bi-Att is the bi-directional attention based parser [24], and NeuroMST is the neural MST parser [95]. "Best Published" includes the most accurate parsers in term of UAS among Koo et al. [77], Martins et al. [105], Martins et al. [106], Lei et al. [85], Zhang et al. [163], Zhang and McDonald [160], Pitler and McDonald [131], and Cheng et al. [24].

performance of our parser is competitive with BIAF, and significantly better than others. The only exception is Japanese, on which NeuroMST obtains the best scores.

5.3.6 Experiments on UD Treebanks

For UD Treebanks, we select 12 languages — Bulgarian, Catalan, Czech, Dutch, English, French, German, Italian, Norwegian, Romanian, Russian and Spanish. For all the languages, we adopt the standard training/dev/test splits, and use the universal POS tags [130] provided in each treebank. The statistics of these corpora are provided in Appendix A.2. For evaluation, we report results excluding punctuation, which is any tokens with POS tags "PUNCT" or "SYM".

	Dev				Test			
	BIAF		STACKPTR		BIAF		STACKPTR	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
bg	93.92±0.13	89.05±0.11	**94.09±0.16**	**89.17±0.14**	94.30±0.16	**90.04±0.16**	**94.31±0.06**	89.96±0.07
ca	94.21±0.05	91.97±0.06	**94.47±0.02**	**92.51±0.05**	94.36±0.06	92.05±0.07	**94.47±0.02**	**92.39±0.02**
cs	94.14±0.03	90.89±0.04	**94.33±0.04**	**91.24±0.05**	94.06±0.04	90.60±0.05	**94.21±0.06**	**90.94±0.07**
de	91.89±0.11	88.39±0.17	**92.26±0.11**	**88.79±0.15**	90.26±0.19	86.11±0.25	**90.26±0.07**	**86.16±0.01**
en	**92.51±0.08**	**90.50±0.07**	92.47±0.03	90.46±0.02	91.91±0.17	89.82±0.16	**91.93±0.07**	**89.83±0.06**
es	93.46±0.05	91.13±0.07	**93.54±0.06**	**91.34±0.05**	93.72±0.07	91.33±0.08	**93.77±0.07**	**91.52±0.07**
fr	**95.05±0.04**	**92.76±0.07**	94.97±0.04	92.57±0.06	92.62±0.15	89.51±0.14	**92.90±0.20**	**89.88±0.23**
it	94.89±0.12	92.58±0.12	**94.93±0.09**	**92.90±0.10**	**94.75±0.12**	**92.72±0.12**	94.70±0.07	92.55±0.09
nl	93.39±0.08	90.90±0.07	**93.94±0.11**	**91.67±0.08**	93.44±0.09	91.04±0.06	**93.98±0.05**	**91.73±0.07**
no	95.44±0.05	93.73±0.05	**95.52±0.08**	**93.80±0.08**	95.28±0.05	93.58±0.05	**95.33±0.03**	**93.62±0.03**
ro	91.97±0.13	85.38±0.03	**92.06±0.08**	**85.58±0.12**	**91.94±0.07**	**85.61±0.13**	91.80±0.11	85.34±0.21
ru	93.81±0.05	91.85±0.06	**94.11±0.07**	**92.29±0.10**	94.40±0.03	92.68±0.04	**94.69±0.04**	**93.07±0.03**

Table 5.4: UAS and LAS on both the development and test datasets of 12 treebanks from UD Treebanks, together with BIAF for comparison.

Table 5.4 summarizes the results of the STACKPTR parser, along with BIAF for comparison, on both the development and test datasets for each language. First, both BIAF and STACKPTR parsers achieve relatively high parsing accuracies on all the 12 languages — all with UAS are higher than 90%. On nine languages — Catalan, Czech, Dutch, English, French, German, Norwegian, Russian and Spanish — STACKPTR outperforms BIAF for both UAS and LAS. On Bulgarian, STACKPTR achieves slightly better UAS while LAS is slightly worse than BIAF. On Italian and Romanian, BIAF obtains marginally better parsing performance than STACKPTR.

5.4 Discussions

In this chapter, we proposed STACKPTR, a transition-based neural network architecture, for dependency parsing. Combining pointer networks with an internal stack to track the status of the top-down, depth-first search in the decoding procedure, the STACKPTR parser is able to capture information from the whole sentence and all the previously derived subtrees, removing the left-to-right restriction in classical transition-based parsers, while maintaining linear parsing steps, w.r.t the length of the sentences. Experimental results on 29 treebanks show the effectiveness of our parser across 20 languages, by achieving state-of-the-art performance on 21 corpora.

There are several potential directions for future work. First, we intend to consider how to conduct experiments to improve the analysis of parsing errors qualitatively and quantitatively. Another interesting direction is to further improve our model by exploring reinforcement learning approaches to learn an optimal order for the children of head words, instead of using a predefined fixed order.

Part II

Interpretability of Neural Structured Prediction Models

Chapter 6

Interpretability of Deep Neural Networks and the Probing Method

The end-to-end training paradigm significantly simplifies the hand-crafted feature engineering process in traditional feature-based machine learning (ML) systems, while giving the neural models flexibility to be optimized towards the ultimate tasks. This simplicity, however, comes at the expense of model interpretability [15, 89, 136]. Unlike traditional feature-engineered NLP systems whose features, e.g. morphological properties, syntactic categories or semantic relations, are more easily understood by humans, it is more difficult to understand what happens in the internal components of an end-to-end neural network. Such deep neural models are sometimes perceived as "black-box", hindering research efforts and limiting their utility to society [13].

The second part of this thesis focuses on model interpretability of deep neural networks. In this chapter, I first review terminologial issues regarding analysis and interpretation in machine learning (Section 6.1. Then I briefly describe the methodological approach of probing, which is used throughout the following chapters for analyzing deep learning models (Section 6.2), and survey related work on applying probing method to analyze neural networks. In the last section (Section 6.3), I apply probes to neural dependency paring models to analyze the part-of-speech (POS) information encoded in their internal representations. According to the experimental results, we argue that, without considering the expressiveness of probing classifiers, the accuracy of probes does not consistently reflect the quality of the encoded information. Based on our findings, we propose to interpret performance of probing tasks with two separate metrics, *capacity* and *accessibility*, which are associated with probe expressiveness.

6.1 Terminological Issues on Interpretability

As discussed in Belinkov [13], terms such as interpretability, explainability, transparency, explainable AI (XAI) have been interchangeably used in the context of work on deep learning, and more broadly machine learning and AI. At present there seems to be no consensus on their precise definition and application to the study of AI systems. I think a short review of aspects of terminology is helpful to clarify the questions this thesis considers in the broader work on interpretability in AI. For detailed discussions, please see [13, 41, 89], as well as the online book by Christoph Molnar[1], for more reflections and references.

Interpretability from explaining decisions. Miller [115] defined interpretability, from the perspective of explanation in social sciences, as “the degree to which an observer can understand the cause of a decision”. In this definition, interpretability is the same as *explainability*. While explaining specific model predictions is obviously important in work on deep learning and is recognized as a desideratum for increasing the accountability of machine learning systems [42], it is different from interpretability on explicitly explaining decisions for given examples.

Interpretability from Transparency Doshi-Velez and Kim [41] defined interpretability as “the ability to explain or to present in understandable terms to human”, not referring to *decisions*. Lipton [89] related interpretability *transparency*, which is concerned with how the model works. One important criterion is the level of analysis. Transparency can operate at the level of the entire model (*simulatability*), at the level of individual components (*decomposability*), and at the level of the training algorithm (*algorithmic transparency*).

Interpretability from post-hoc explanations Lipton [89] also contrasts transparency with *post-hoc explanation*, which is what else can the model tell us by extracting information from

[1]`https://christophm.github.io/interpretable-ml-book/`

learned models. Some common approaches to post-hoc explanations include natural language explanations, visualizations of learned representations or models, and probing the internal layers of deep neural networks.

This thesis aims to use probing methods to provide a better understanding of the learning properties of different parts and modules in deep learning models (striving for decomposibility and algorithmic transparency, in the sense of [89]).

6.2 Methodology of Probing

The key idea of probing method is to utilize supervised learning algorithms to probe internal representations in end-to-end neural models, to predict linguistic properties, such as part-of-speech or morphology. Specifically, the method consists of three steps:

1. train an end-to-end model on a complex task, such as machine translation.
2. use the trained model to generate feature representations of different layers.
3. train a classifier using the generated features to make predictions for a relevant auxiliary task, such as POS tagging.

This process is illustrated in Figure 6.1

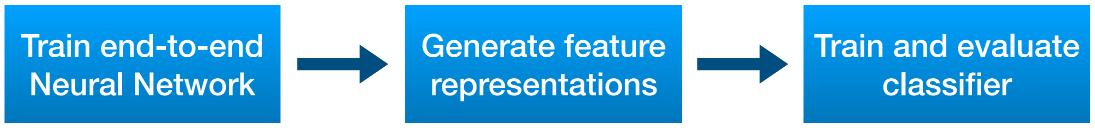

Train end-to-end Neural Network
Generate feature representations
Train and evaluate classifier

Figure 6.1: Framework of the methodology of probing method.

In their pioneering work, Ettinger et al. [47] and Shi et al. [136] investigated intermediate layers of deep neural models in NLP and Alain and Bengio [4] in computer vision. Both of them used linear classifiers as their probes. Subsequently, extensive applications of probing method have been explored. Adi et al. [2] and Conneau et al. [32] conducted probing tasks to study the linguistic properties of sentence embedding methods. Liu et al. [90] and Tenney et al. [146] focused on contextual word representations, evaluating CoVe [107], ELMo [129] and BERT [39] on a variety of linguistic tasks. These research attempted to answer the natural question: *what*

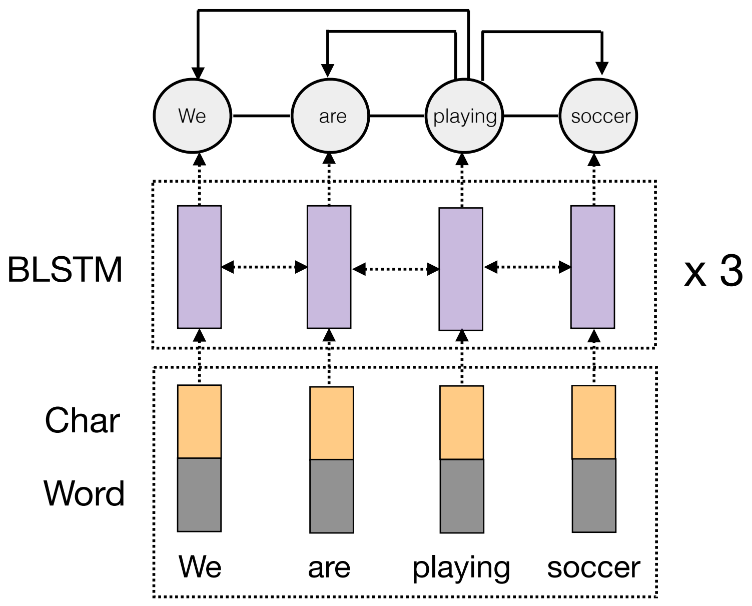


Figure 6.2: Architecture of the BLSTM-CNN encoder in the three neural parsers. The "char" embedding is the output representation from the CNN layer.

linguistic information is captured in the internal representations of neural networks, with the basic assumption that the performance of the probes reflects the quality of representations [13]. Commonly used probes includes linear classifiers and multiple layer perceptron (MLP) with one or two hidden layers [65, 90]. However, does the accuracy of these commonly used probes accurately reflect the quantity of the information encoded in the representations? Furthermore, is it sufficient to characterize the information in a representation using a single accuracy measure?

Previous work has examined how the choice of probing tasks and models, and comparing baselines affect the probing conclusion. Some studies [14, 66, 146] used non-contextual word embeddings or models with random weights as baselines. Zhang and Bowman [161] presented experiments for understanding the roles probe training sample size have on linguistic task accuracy. Hewitt and Liang [65] designed control tasks to explore the relationship between representations, probes and task accuracies. However, almost all these previous studies (to our best knowledge) utilized linear classifiers or MLP with one or two hidden layers as their probes, without comparing with more expressive classifiers.

In next section, we use probing methods to investigate the to analyze the part-of-speech (POS) information encoded in the internal representation of three state-of-the-art neural dependency parsing models — Deep Biaffine Parser, NeuroMST parser and Stack-Pointer Parser, and demonstrate the importance of associating probing accuracy with classifier expressiveness.

6.3 Probing POS Information in Neural Dependency Parsers

In this section, we first explore the consistency of using probes with different expressiveness: *Do probing classifiers with different expressiveness lead to consistent observations?* Then we conduct a battery of experiments to investigate the POS information in neural dependency parsers by answering the following questions:

- What is the division of labor between word and character embeddings?
- Which parts of the neural architecture capture POS information?
- Does the addition of accurate POS information impact the learned representations in terms of POS information?
- What impact does the choice of parsing algorithms (graph-based vs. transition-based) have on the learned representations?

All experiments are performed on PTB with the same settings in Ma et al. [101].

We select dependency parsing as the test bed primarily because of the accessibility to annotated data across a broad spectrum of languages [121] and the sufficiently sophisticated performance of state-of-the-art dependency parsers [43, 95, 101]. we conduct probing experiments on three neural dependency parsing models — Deep Biaffine Parser [43], NeuroMST parser [95] (in Chapter 4 and Stack-Pointer Parser [101] (in Chapter 5). All the three parsing models are implemented with the bi-directional LSTM-CNN architecture (BLSTM-CNN) [26, 94] as encoder to incorporate both word-level and character-level information[2]. The BLSTM-CNN encoder consists of three bi-directional LSTM layers (see the architecture depicted in Figure 6.2).

[2]For Deep Biaffine parser we use the re-implemented version in Ma et al. [101]

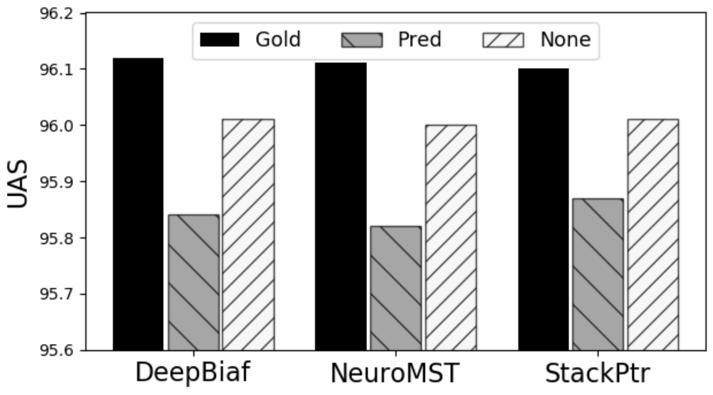


Figure 6.3: UAS on the test data of PTB with different versions of POS tags.

6.3.1 Motivation: a case study on the effect of POS embedding

From the results in Section 5.3.4 we observed that end-to-end neural parsers without POS information can obtain even better performance than parsers using imperfect predicted POS tags. To verify this, we performed similar experiments on English Penn Treebank (PTB) [104] using the three parsing models. For each parsing model, we evaluate the parser with gold-standard and predicted POS[3] tags, and without POS tags, respectively.

Figure 6.3 lists the unlabeled attachment score (UAS) of the three parsers with different versions of POS tags on PTB. Consistent with the observation in Ma et al. [101], the parsers with gold-standard POS tags achieve the best UAS, while the parsers with predicted (imperfect) POS tags perform slightly worse than the parsers that do not use explicit POS tags at all. This observation raises a question: how much POS information is captured implicitly (if anything) by neural parsing models and where this information is stored, which motivates us to investigate the information encoded in the internal representations of these parsers.

6.3.2 Word and Character Embeddings

We evaluate the POS tagging performance with three probing classifiers — linear logistic regression (Linear), multiple layer perceptron with one hidden layer (MLP) and Support Vector Machine with RBF kernel (SVM-RBF) — on the word and character embeddings. For comprehensive

[3]For predicted POS tags, we used a pretrained POS tagger with 97.3% accuracy.

analysis, we also evaluate the representation that concatenates word and character embeddings (W$\oplus$C).

Table 6.1 shows the probing results with three classifiers, together with MFT, which is the most frequent tag baseline on word types[4]. For each classifier, we present the accuracy of both the POS tagging and its control task [65], which associates word types with random outputs to complement linguistic tasks, together with the corresponding selectivity which is the gap between the accuracies of the linguistic task and its control task. Note that MFT can be regarded as an upper bound for non-contextual representations, such as word and character embeddings, for both POS tagging and its control task. The observations of Linear and MLP classifiers on the three representations are similar, with the difference on the selectivity of the two probes. According to the performance of Linear and MLP classifiers, the word representation encodes more POS information than the character one, while the concatenated representation obtains much better accuracy, even better than the MFT upper bound. If we only consider those results from Linear and MLP classifiers, we may conclude that the POS information encoded in word and character embeddings are complementary to each other. However, the performance of SVM-RBF classifier provides significantly different observations: (i) the character embedding achieves better accuracy than the word embedding (93.5% vs. 92.8%); (ii) the concatenated representation achieves similar accuracy with the character embedding (93.7% vs. 93.5%), leading to the conclusion that POS information in the word embedding is mostly covered by the character embedding.

6.3.3 Capacity vs. Accessibility

The inconsistent observations of probing classifiers with different expressiveness lead us to re-think about how to interpret performance of probing tasks. We cannot meaningfully compare the linguistic properties of internal representations of neural networks using only linguistic task accuracy. As long as one representation is a lossless mapping of the other one, a sufficiently expressive probe with sufficient amount of training data can obtain the same accuracy on top of them. However, it fails to take into account differences in ease of memorization between them. In Table 6.1, for example, with SVM the character embedding encodes more POS information than

[4]For out-of-vocabulary (OOV) words, we assign them the most common POS tags.

	Linear			MLP			SVM-RBF			MFT		
Layers	POS	Ctl.	Sel.	POS	Ctl.	Sel.	POS	Ctl.	Sel.	POS	Ctl	Sel.
Word	91.9	58.0	33.9	92.0	65.1	26.9	92.8	94.0	-1.2	92.6	96.0	-3.4
Char	90.3	58.3	32.0	90.5	70.6	19.9	93.5	96.1	-2.6	–	–	–
W⊕C	93.4	65.9	27.5	93.5	74.8	18.7	93.7	96.2	-2.5	–	–	–

Table 6.1: Performance of three probing classifiers on word and character embeddings, and their concatenation of a DeepBiaf parser trained on PTB without POS tags as input. MFT is the most frequent tag baseline. Clt. is the perfomance of the POS tagging control task and Sel. is the corresponding selectivity.

the word embedding. But the POS information in the word embedding is easier to be detected by probes — a simple linear probing classifier obtains higher accuracy on top of the word embedding.

In this work, we propose to analyze representations with two separate metrics that are associated with the expressiveness of the probe families:

Capacity: *the linguistic task accuracy of sufficiently expressive probes.*

Accessibility: *the linguistic task accuracy of probes in linear model family.*

As two separate metrics, *capacity* measures how much information of the linguistic knowledge has been encoded in the representation, while *accessibility* measures how easily the information can be detected by a probe. In the rest of this paper, we use SVM-RBF as the probe to approximate capacity and linear logistic regression for accessibility.

The probe selectivity associated with control tasks [65], which is a metric of the probe, gives us indirect intuition for the ease of information memorization across different representations. Different from it, our proposed two metrics provide direct insight into how much information has been encoded and how easily the information can be accessed by ML models.

6.3.4 Effects of POS Embedding as Input

Table 6.2 illustrates the accessibility and capacity on representations from different layers of DeepBiaf parsers trained w./wo. POS tags as input. We observe that POS embedding has

	Word		Char		W⊕C		LSTM 1		LSTM 2		LSTM 3	
POS	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.
WO.	91.9	92.8	**90.3**	93.5	93.4	93.7	97.5	97.7	97.4	97.8	95.7	96.8
W.	91.8	92.8	**89.5**	93.5	93.5	93.7	100.0	100.0	99.7	99.7	97.3	98.4

Table 6.2: Accessibility (Acces.) and Capacity (Cap.) on representations from different layers of DeepBiaf parsers trained on PTB w./wo. POS tags as input. Qualities on character embeddings are highlighted because of the significant difference.

no significant impact on word or character embeddings, except the accessibility of character embedding — without using POS embedding the accessibility of character embedding on POS information improves.

We also observe that even without using POS embedding, the LSTM layers of the encoder capture a large amount of POS information. Both the accessibility and capacity of the first and second LSTM layers are better than the accuracy (97.3%) of the predicted POS tags, explaining why the parsers without using POS tags achieved better performance than those using predicted POS tags. In fact, the POS tagging accuracy of 97.8% (the capacity of the second LSTM layer) is competitive or even better than the state-of-the-art neural POS tagging systems [35, 94]. It verifies that learning a more complex task (dependency parsing) benefits the simpler upstream tasks (POS tagging).

Another interesting observation is that, for both parsers trained w./wo. POS embedding, the POS information captured in the top LSTM layer (LSTM 3) declines in both accessibility and capacity comparing with the first two LSTM layers. One possible explanation for the discrepancy is that the top LSTM layer, which is directly used for the parsing, pays more attention to parsing task.

6.3.5 Effects of Parsing Algorithms

Table 6.3 presents the results on representations from layers of three dependency parsers. For the two graph-based parsers with different training objectives (edge-factored loss of DeepBiaf vs. global structured loss of NeuroMST), there is no significant difference on either the accessibility or

	Word		Char		W⊕C		LSTM 1		LSTM 2		LSTM 3	
Parsers	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.	Acces.	Cap.
DeepBiaf	91.9	92.8	90.3	93.5	93.4	93.7	97.5	97.7	97.4	97.8	95.7	96.8
NeuroMST	91.9	92.8	90.1	93.5	93.4	93.7	97.5	97.6	97.4	97.8	96.0	96.9
StackPtr	91.8	92.9	90.1	93.5	93.4	93.7	97.5	97.7	97.1	97.7	**94.0**	**96.1**

Table 6.3: Accessibility (Acces.) and Capacity (Cap.) on representations from the layers of three parsers. Results that are significantly different from that of the same layers of other parsers are highlighted.

the capacity of the representations.

When comparing the transition-based StackPtr with the two graph-based parsers (DeepBiaf and NeuroMST), we observe that for non-contextual representations (word and character embeddings) and the first two LSTM layers, the capacity and accessibility are similar. But when the depth increases, both the capacity and accessibility of the POS information in the transition-based StackPtr parser declines more rapidly than the two graph-based parsers, demonstrating that different decoding algorithms indeed impact the output representations of the encoder.

6.4 Discussions

In this chapter, we probed the internal representations of three neural dependency parsers. Through analyzing the results of probes in different expressive families, we found that these probes obtained significant different accuracies. Based on this, we proposed tow complementary metrics: Capacity and Accessibility, to enhance the interpretation of results from probing methods.

In the next chapter, we will introduce how to use probes to investigate learning properties of deep neural networks, rather than analyzing what linguistic information is captured in their internal layers.

Chapter 7

Probing Learning Properties of Neural Networks

As discussed in the previous chapter, the research of probing method attempted to answer the natural question: what linguistic information is captured in the internal representations of neural networks. Moreover, besides using probes to analyze what linguistic information is captured in neural networks, can we use probes to understand other properties of deep learning models? These questions still remain unclear and no attempt (to our best knowledge) has been made to answer them.

In this chapter, we use probing methods to investigate the learning properties of deep neural networks with dependency parsing as a test bed. By conducting systematic experiments, we illustrate two learning properties of deep neural networks: (i) *laziness* — modules of a neural network will not actively learn information that is already learned by other modules (Section 7.1); (ii) *targetedness* — information, if unnecessary for the target task, will be filtered out from the internal representations (Section 7.2).

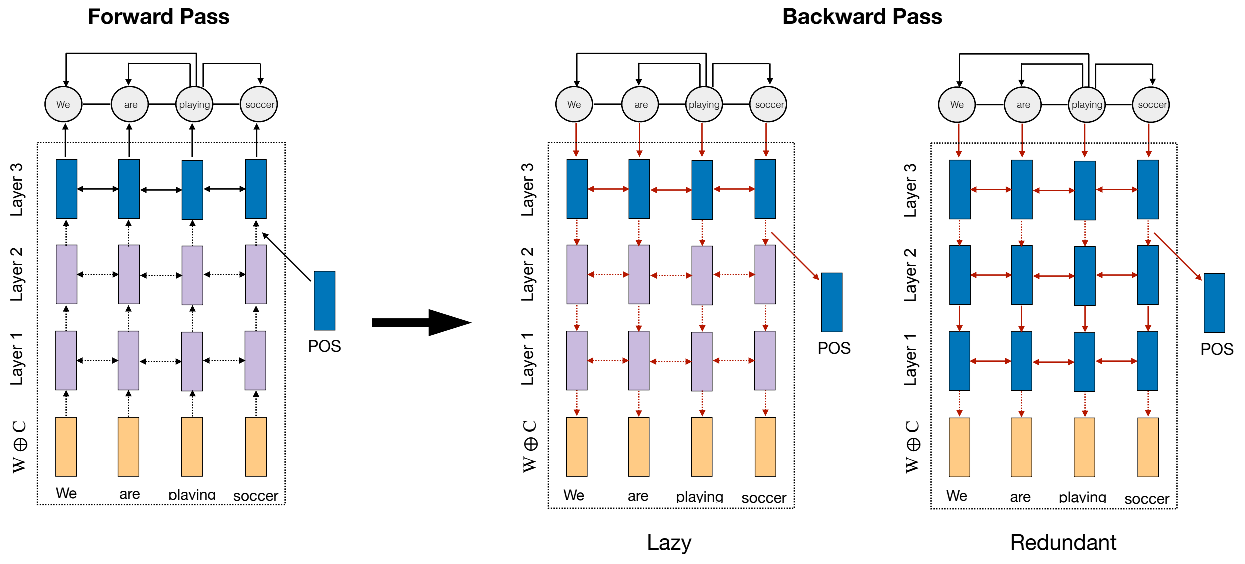


Figure 7.1: An example for the illustration of our designed experiments to examine the laziness of deep neural networks.

7.1 Laziness: Information Propagation through Deep Neural Networks

In this section, we investigate one learning property of neural networks by analyzing the information propagation during neural network training. Results from carefully designed probing experiments illustrate that deep neural networks exhibit the learning property of *laziness* — information, if already encoded in some components of a neural network, will not be propagated to the network's other components.

7.1.1 Laziness vs. Redundancy

In this work, laziness, or the opposite redundancy, is defined on the pattern of how neural networks distribute information across their components — do they store one kind of information/knowledge in a single module or multiple ones?

As discussed in the pioneering of Dropout [139], redundancy is commonly a desired property and one of the primary motivations of the dropout approach. Redundancy is closely related to robustness, because a system that stores redundant information has better chance to work robustly

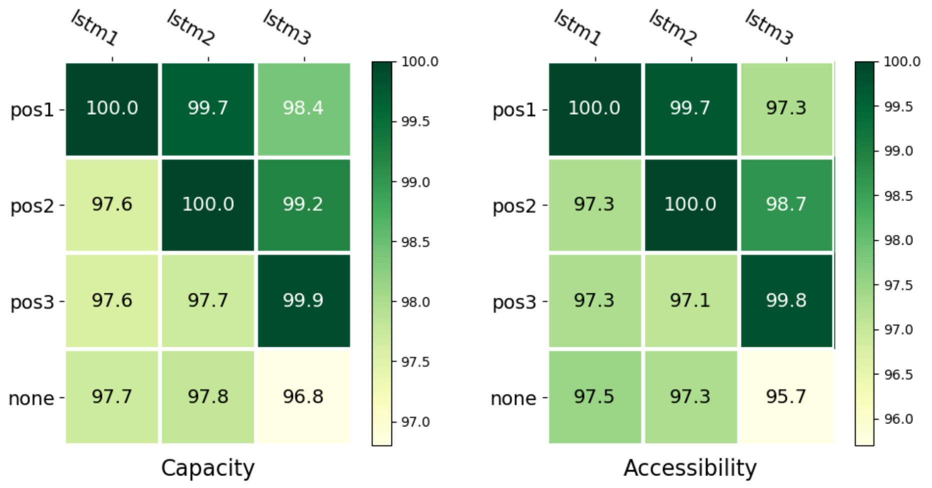


Figure 7.2: The heatmap of capacity and accessibility of DeepBiaf parsers with POS tags as input to different layers. pos1, pos2 and pos3 refer to the parser with gold-standard POS tags as input to the 1st, 2nd and 3rd LSTM layers. none indicates the parser without POS tags as input.

under non-stationary conditions. Laziness, on the other hand, is probably the way that requires minimal amount of efforts to learn knowledge or to fullfil target tasks. Dividing the target task into smaller individual sub-tasks, though might not be robust, may be the most efficient way to complete the ultimate goal.

7.1.2 Experiment Design

The key idea of the experiments is to examine the learning behaviors of neural parsers by placing the gold-standard POS tags as input to different encoder layers. Suppose that the gold-standard POS tags are fed as the input of the third LSTM layer (see Figure 7.1 for illustration). Then, during the forward pass to compute the parsing objective, the gold-standard POS information will be passed to the third LSTM layer and the layers on top of it, but not the layers below it. During the backward pass in training, however, the gradients that might carry gold-standard POS information will be back-propagated to the entire neural network. By probing the layers of this neural parser, we can examine if the gold-standard POS information has been stored in the layers below the one that gold-standard POS tags are fed into or not. If the gold-standard POS information has been propagated to the lower layers, as illustrated in the second case in the right side of Figure 7.1, the POS accuracy of probes should be significantly better than the one without using POS tags, showing the redundancy. Otherwise, it demonstrates the laziness.

In this experiment, we train DeepBiaf parsers on PTB with gold-standard POS tags as input to different LSTM layers. Dropout [139] is applied everywhere (see Ma et al. [101] for details). Figure 7.2 shows the heatmap of the capacity and accessibility. From the heatmap, we can see clearly that the probing accuracies in the layers below the one that POS tags are fed into are significantly worse than that of the above layers. It implies that the POS information has not been propagated into those lower layers through back propagation, verifying the laziness of neural networks. Importantly, dropout, which is introduced to enhance the redundancy of neural networks, does not as expected prevent the laziness learning of the neural parsers in our experiments. What the real impact dropout has on neural network training remains an open problem and might be an interesting direction for future work [100].

7.2 Targetedness: Semantic Information in Syntactic Parsers

In this section, we explore another learning property of neural networks, *targetedness*, by asking the question how deep neural networks process information that is unnecessary for their ultimate tasks — will this information be retained in their internal layers or filtered out?

7.2.1 Lexical Semantic Tagging

To answer the above question, we conduct experiments to probe neural dependency parsers with lexical semantic tagging (SEM) [19] as the linguistic probing task. SEM is a sequence labeling task: given a sentence, the goal is to assign to each word a tag representing a semantic class. Figure 7.3 gives an example of SEM. Linguistically, some lexical semantic information is unnecessary for syntactic parsing. For instance, w.r.t dependency parsing, the semantic tag PER (for person) of the token "Tom" provides no more useful information than its POS tag NNP (for proper noun). With SEM as the probing task, we want to examine if these unnecessary lexical semantic information will be filtered out from dependency parsing models trained with only syntactic supervision.

SEM	PER	NOW	NOT	EXS	QUV	CON	.
POS	NNP	VBZ	RB	VB	JJ	NN	.
	Tom	does	not	drink	much	beer	.

Figure 7.3: An example of lexical semantic tagging.

For the annotated data of lexical semantic tagging, we use the Groningen Parallel Meaning Bank (PMB) [1] which includes 66 fine-grained semantic tags grouped in 13 coarse categories. The experiments are conducted on the silver part of the dataset — we randomly split the data into training, validation and test sets with the proportion of $[8, 1, 1]$.

7.2.2 Baselines

In order to analyze the lexical semantic information memorized in the neural networks, we need reasonable baselines for comparison. In Belinkov [13], the authors proposed two baselines: (i) assigning to each word the most frequent tag (MFT) according to the training data, with the global majority tag for unseen words; (ii) training with unsupervised word embeddings as features for the classifier. However, these two baselines are arguably unsuitable for our experiments, since they are based on non-contextual features. Comparing these two baselines with the internal contextual representations in neural dependency parsers is unfair and cannot lead to reliable conclusions. In other words, even if the probes on the internal representations achieve significant better accuracy than the two baselines, we cannot conclude that the neural dependency parser has learned lexical semantics. The reason is that neural parsers are able to learn POS information (as shown in Section 6.3), which is highly correlated with lexical semantic information (assigning to each word the most frequent tag based on its POS tag obtains 75.2% accuracy). Thus, to claim that the neural parsers cant capture lexical semantic information, we need to proof that the representations capture more information than that provided by POS tags.

In this paper, we propose two new baselines: (i) assigning to each word the most frequent SEM tag based on the combination of its word type and POS tag; (ii) a linear classifiers trained on binary features of word type of each token and context of POS tags in a small neighborhood

		SEM Accuracy	
MFT (word)		85.0	
MFT (word + POS)		89.9	
BF-Linear		**93.4**	
BLSTM-CNN-CRF		**94.9**	
		Acces.	Cap.
Layers	Word	82.3	83.8
	Char	80.3	85.2
	W$\oplus$C	85.4	85.3
	LSTM 1	91.5	**91.7**
	LSTM 2	90.1	91.6
	LSTM 3	86.8	89.5

Table 7.1: Accessibility and Capacity of lexical semantic tagging on representations from layers of DeepBiaf dependency parsers, together with MFT baselines, the linear classifier on binary features (BF-Linear) and the upper bound accuracy of the sequence labeling model.

(window of 3). The POS tags are automatically labeled with Standford POS Tagger[1]. Note that neither of these two baselines utilize contextual information beyond POS tags. We also include an upper bound of training a BLSTM-CNN-CRF sequence labeling model [94] for SEM tagging.

7.2.3 Experimental Results

The experiments are conducted on DeepBiaf parsers trained on PTB without using POS tags as input. Table 7.1 summarizes the results of training probes to predict SEM tags using representations generated by different encoding layers of the DeepBiaf parser. Comparing representations from LSTM layers 1 through 3, SEM tagging accuracy peaks at layer 1 and does not improve at higher layers. The best capacity score from LSTM layer 1 is 91.7%, slightly worse than the baseline of linear classifier with binary features, far below the upper bound from the sequence labeling model.

[1]`https://nlp.stanford.edu/software/tagger.shtml`

It indicates that the internal representations do not learn more contextual information that benefits SEM prediction than POS tags.

7.2.4 Discussion

Extensive studies investigated the requested semantic information in syntactic parsing, such as selectional restrictions [8, 71, 72, 155]. Lexical semantic categories, in some cases, are indeed relevant for selectional restrictions and in turn can be used for disambiguation in syntactic parsing. One possible reason that neural dependency parsers are not able to learn relevant semantic information is that there are few such cases in the training data. Another possible reason might be due to the SEM data. The PMB data used in our experiments do not support analysis on specific case studies, but only an average accuracy on all the semantic categories. Fine-grained investigation on specific categories of semantic information learned by neural syntactic parsers might be an interesting direction for future work.

7.3 Conclusion

In this chapter, using the the two metrics proposed in Section 6.3.3, we have conducted experiments to investigate the learning properties of neural networks. Experimental results illustrate two learning behaviors of neural networks: (i) laziness — modules of a neural network will not actively learn information that is already learned by other modules; (ii) targetedness — information that is unnecessary for the ultimate objective will be filtered out.

Chapter 8

Conclusions and Future Work

8.1 Conclusion

In this dissertation, we gave readers a thorough overview of neural networks for linguistic structured prediction: three neural models for sequence and tree structured prediction tasks (PART I) and the interpretability of neural dependency parsing models (PART II).

In Chapter 2, we walked through some background of linguistic structured prediction, end-to-end learning paradigm and give an overview of the history and recent development of neural representation learning on linguistic structured prediction. In the last section of this chapter, we proposed a consistent neural architecture, named BLSTM-CNNs, for the encoding component (encoder) across different structured prediction tasks.

In Chapter 3, 4 and 5, we applied BLSTM-CNNs to sequence labeling and dependency parsing tasks, by by stacking different structured decoding layers on top of the BLSTM-CNNs encoder. Through experiments on different tasks, corpus and languages, our neural structured prediction models achieved or near the state-of-the-art performance.

In Chapter 6, we briefly discussed the terminological issues regarding analysis and interpretation in machine learning. Then we revisited the probing method [13, 47, 136], and appled it to investigate how part-of-speech information are memorized in neural dependency parsers.

In Chapter 7, we conducted systematic experiments to illustrate two learning properties of deep neural networks: (i) *laziness* – modules of a neural network will not actively learn information that is already learned by other modules; (ii) *targetedness* – information, if unnecessary for the end task, will be filtered out from the internal representations.

8.2 Future Work

This thesis opens up several questions for future research.

8.2.1 Encoding Structured Dependencies in Representations: No Structured Algorithms in Structured Predictions

Although the neural networks proposed in this thesis obtained outstanding performance for a wide range of tasks and languages, they still suffer some problems: (i) the design and combination of the structured output layers with the end-to-end neural representation encoders is not easy. Every step of the structured algorithms must be ensured differentiable so that the gradients can be back-propagated to the entire network to achieve end-to-end learning; (ii) most of the decoding algorithms are inefficient in practice because they are not parallelizable.

To fundamentally solve these problems, one potential direction is to encode the underlying dependencies of the structured outputs into intermediate representations to get rid of structured training and decoding algorithms. Ma et al. [102] proposed to a non-autoregressive sequence generation model to avoid sequential decoding algorithm. On direction of future research is to extend this framework to general structured prediction tasks.

8.2.2 Investigating Information Encoding Schema in Representations

Another interesting direction of future research is to investigate the information encoding schema of different neural architectures. From the observations in our probing experiments, different representations, even encoding the same knowledge, may encode them in different format. The different information encoding schema is highly correlated with the learning properties of the neural architecture. Therefore, figuring out the relation between encoding schema and the neural architectures is useful for us to better understand different neural architectures.

8.2.3 Inductive Bias from Learning Properties of Different Neural Architectures

Finally, on promising direction of future research is to explore useful inductive bias from investigating the learning properties of different neural architectures. By investigating the learning properties, such as the two learning properties of we investigated in Chapter 7, we can conclude architectural inductive biases and attempt to apply them to advanced representation learning, such as disentanglement, to get rid of explicit supervision.

Appendix A

Dependency Parsing Experiments

A.1 Hyper-parameters

A.1.1 NeuroMST Parser

Table A.1 summarizes the chosen hyper-parameters for NeuroMST parser. We tune the hyper-parameters on the development sets by random search. Due to time constrains it is infeasible to do a random search across the full hyper-parameter space. Thus, we use the same hyper-parameters across the models on different treebanks and languages. It also demonstrates the robustness of our parsing model. Note that we use 2-layer BLSTM followed with 1-layer MLP. We set the state size of LSTM to 256 and the dimension of MLP to 100. Tuning these two parameters did not significantly impact the performance of our model.

Layer	**Hyper-parameter**	**Value**
CNN	window size	3
	number of filters	50
LSTM	number of layers	2
	state size	256
	initial state	0.0
	peepholes	Hadamard
MLP	number of layers	1
	dimension	100
Dropout	embeddings	0.15
	LSTM hidden states	0.25
	LSTM layers	0.33
Learning	optimizer	Adam
	initial learning rate	0.002
	decay rate	0.5
	gradient clipping	5.0

Table A.1: Hyper-parameters for NeuroMST parser.

Layer	Hyper-parameter	Value
CNN	window size	3
	number of filters	50
LSTM	encoder layers	3
	encoder size	512
	decoder layers	1
	decoder size	512
MLP	arc MLP size	512
	label MLP size	128
Dropout	embeddings	0.33
	LSTM hidden states	0.33
	LSTM layers	0.33
Learning	optimizer	Adam
	initial learning rate	0.001
	(β_1, β_2)	(0.9, 0.9)
	decay rate	0.75
	gradient clipping	5.0

Table A.2: Hyper-parameters for StackPtr parser.

A.1.2 StackPtr Parser

Table A.2 summarizes the chosen hyper-parameters used for all the experiments in this paper. Some parameters are chosen directly or similarly from those reported in Dozat and Manning [43]. We use the same hyper-parameters across the models on different treebanks and languages, due to time constraints.

A.2 UD Treebanks

The statistics of these corpora are provided in Table A.3.

	Corpora		*#Sent*	*#Token (w.o punct)*
		Training	8,907	124,336 (106,813)
Bulgarian	BTB	Dev	1,115	16,089 (13,822)
		Test	1,116	15,724 (13,456)
		Training	13,123	417,587 (371,981)
Catalan	AnCora	Dev	1,709	56,482 (50,452)
		Test	1,846	57,738 (51,324)
	PDT, CAC	Training	102,993	1,806,230 (1,542,805)
Czech	CLTT	Dev	11,311	191,679 (163,387)
	FicTree	Test	12,203	205,597 (174,771)
	Alpino	Training	18,310	267,289 (234,104)
Dutch	LassySmall	Dev	1,518	22,091 (19,042)
		Test	1,396	21,126 (18,310)
		Training	12,543	204,585 (180,308)
English	EWT	Dev	2,002	25,148 (21,998)
		Test	2,077	25,096 (21,898)
		Training	14,554	356,638 (316,780)
French	GSD	Dev	1,478	35,768 (31,896)
		Test	416	10,020 (8,795)
		Training	13,841	263,536 (229,204)
German	GSD	Dev	799	12,348 (10,727)
		Test	977	16,268 (13,929)
		Training	12,838	270,703 (239,836)
Italian	ISDT	Dev	564	11,908 (10,490)
		Test	482	10,417 (9,237)
	Bokmaal	Training	29,870	48,9217 (43,2597)
Norwegian	Nynorsk	Dev	4,300	67,619 (59,784)
		Test	3,450	54,739 (48,588)
		Training	8,043	185,113 (161,429)
Romanian	RRT	Dev	752	17,074 (14,851)
		Test	729	16,324 (14,241)
		Training	48,814	870,034 (711,184)
Russian	SynTagRus	Dev	6,584	118,426 (95,676)
		Test	6,491	117,276 (95,745)
	GSD	Training	28,492	827,053 (730,062)
Spanish	AnCora	Dev	4,300	89,487 (78,951)
		Test	2,174	64,617 (56,973)

Table A.3: Corpora statistics of UD Treebanks for 12 languages. *#Sent* and *#Token* refer to the number of sentences and the number of words (w./w.o punctuations) in each data set, respectively.

A.3 Detailed Results of Stack-Pointer Parser

Table A.4 illustrates the details of the experimental results. For each STACKPRT parsing model, we ran experiments with decoding beam size equals to 1, 5, and 10. For each experiment, we report the mean values with corresponding standard deviations over 5 runs.

		English							
		Dev				**Test**			
Model	beam	UAS	LAS	UCM	LCM	UAS	LAS	UCM	LCM
BiAF	–	95.73±0.04	**93.97±0.06**	60.58±0.77	47.47±0.63	95.84±0.06	**94.21±0.04**	59.49±0.23	49.07±0.34
Basic	1	95.71±0.02	93.88±0.03	62.33±0.33	47.75±0.32	95.71±0.06	94.07±0.06	60.91±0.35	49.54±0.48
	5	95.71±0.04	93.88±0.05	62.40±0.45	47.80±0.44	95.76±0.11	94.12±0.11	61.09±0.43	49.67±0.41
	10	95.72±0.03	93.89±0.04	**62.40±0.45**	**47.80±0.44**	95.77±0.11	94.12±0.11	61.09±0.43	49.67±0.41
+gpar	1	95.68±0.04	93.82±0.02	61.82±0.36	47.32±0.14	95.73±0.04	94.07±0.05	60.99±0.46	49.83±0.59
	5	95.67±0.01	93.83±0.02	61.93±0.32	47.44±0.20	95.76±0.06	94.11±0.06	61.23±0.47	50.07±0.59
	10	95.69±0.02	93.83±0.02	61.95±0.32	47.44±0.20	95.78±0.05	94.12±0.06	61.24±0.46	**50.07±0.59**
+sib	1	95.75±0.03	93.93±0.04	61.93±0.49	47.66±0.48	95.77±0.15	94.11±0.06	61.32±0.37	49.75±0.29
	5	95.74±0.02	93.93±0.05	62.16±0.22	47.68±0.54	95.84±0.09	94.17±0.09	61.52±0.57	49.91±0.76
	10	95.75±0.02	93.94±0.06	62.17±0.20	47.68±0.54	95.85±0.10	94.18±0.09	**61.52±0.57**	49.91±0.76
Full	1	95.63±0.08	93.78±0.08	61.56±0.63	47.12±0.36	95.79±0.06	94.11±0.06	61.02±0.31	49.45±0.23
	5	95.75±0.06	93.90±0.08	62.06±0.42	47.43±0.36	95.87±0.04	94.20±0.03	61.43±0.49	49.68±0.47
	10	**95.75±0.06**	93.90±0.08	62.08±0.39	47.43±0.36	**95.87±0.04**	94.19±0.04	61.43±0.49	49.68±0.47
		Chinese							
		Dev				**Test**			
Model	beam	UAS	LAS	UCM	LCM	UAS	LAS	UCM	LCM
BiAF	–	90.20±0.17	88.94±0.13	43.41±0.83	38.42±0.79	90.43±0.08	89.14±0.09	42.92±0.29	38.68±0.25
Basic	1	89.76±0.32	88.44±0.28	45.18±0.80	40.13±0.63	90.04±0.32	88.74±0.40	45.00±0.47	40.12±0.42
	5	89.97±0.13	88.67±0.14	45.33±0.58	40.25±0.65	90.46±0.15	89.17±0.18	45.41±0.48	40.53±0.48
	10	89.97±0.14	88.68±0.14	45.33±0.58	40.25±0.65	90.48±0.11	89.19±0.15	45.44±0.44	40.56±0.43
+gpar	1	90.05±0.14	88.71±0.16	45.63±0.52	40.45±0.61	90.28±0.10	88.96±0.10	45.26±0.59	40.38±0.43
	5	90.17±0.14	88.85±0.13	46.03±0.53	40.69±0.55	90.45±0.15	89.14±0.14	45.71±0.46	40.80±0.26
	10	90.18±0.16	88.87±0.14	**46.05±0.58**	40.69±0.55	90.46±0.16	89.16±0.15	45.71±0.46	40.80±0.26
+sib	1	89.91±0.07	88.59±0.10	45.50±0.50	40.40±0.48	90.25±0.10	88.94±0.12	45.42±0.52	40.54±0.69
	5	89.99±0.05	88.70±0.09	45.55±0.36	40.37±0.14	90.41±0.07	89.12±0.07	45.76±0.46	40.69±0.52
	10	90.00±0.04	88.72±0.09	45.58±0.32	40.37±0.14	90.43±0.09	89.15±0.10	45.75±0.44	40.68±0.50
Full	1	90.21±0.15	88.85±0.15	45.83±0.52	40.54±0.60	90.36±0.16	89.05±0.15	45.60±0.33	40.73±0.23
	5	90.23±0.13	88.89±0.14	46.00±0.54	**40.75±0.64**	90.58±0.12	89.27±0.11	**46.20±0.26**	**41.25±0.22**
	10	**90.29±0.13**	**88.95±0.13**	46.03±0.54	**40.75±0.64**	**90.59±0.12**	**89.29±0.11**	**46.20±0.26**	**41.25±0.22**
		German							
		Dev				**Test**			
Model	beam	UAS	LAS	UCM	LCM	UAS	LAS	UCM	LCM
BiAF	–	**93.60±0.13**	**91.96±0.13**	58.79±0.25	49.59±0.19	**93.85±0.07**	**92.32±0.06**	60.60±0.38	52.46±0.24
Basic	1	93.35±0.14	91.58±0.17	59.64±0.78	49.75±0.64	93.39±0.09	91.85±0.09	61.08±0.31	52.21±0.53
	5	93.49±0.14	91.72±0.16	59.99±0.69	49.82±0.54	93.61±0.09	92.07±0.08	61.38±0.30	52.51±0.43
	10	93.48±0.14	91.71±0.17	**60.02±0.69**	49.84±0.54	93.59±0.09	92.06±0.08	61.38±0.30	52.51±0.43
+gpar	1	93.39±0.07	91.66±0.13	59.59±0.54	49.81±0.42	93.44±0.07	91.91±0.11	61.73±0.47	52.84±0.48
	5	93.47±0.09	91.75±0.10	59.81±0.55	50.05±0.39	93.68±0.04	92.16±0.04	62.09±0.44	53.13±0.42
	10	93.48±0.08	91.76±0.09	59.89±0.59	50.09±0.40	93.68±0.05	92.16±0.03	62.10±0.42	**53.14±0.4**
+sib	1	93.43±0.07	91.73±0.08	59.68±0.25	49.93±0.30	93.55±0.07	92.00±0.08	61.90±0.50	52.79±0.22
	5	93.53±0.05	91.83±0.07	59.95±0.23	50.14±0.39	93.75±0.09	92.20±0.08	62.21±0.38	53.03±0.18
	10	93.55±0.06	91.84±0.07	59.96±0.24	**50.15±0.40**	93.76±0.09	92.21±0.08	**62.21±0.38**	53.03±0.18
Full	1	93.33±0.13	91.60±0.16	59.78±0.32	49.78±0.29	93.50±0.04	91.91±0.11	61.80±0.28	52.95±0.37
	5	93.42±0.11	91.69±0.12	59.90±0.27	49.94±0.35	93.64±0.03	92.10±0.06	61.89±0.21	53.06±0.36
	10	93.40±0.11	91.67±0.12	59.90±0.27	49.94±0.35	93.64±0.03	92.11±0.05	61.89±0.21	53.06±0.36

Table A.4: Parsing performance of different variations of our model on both the development and

Bibliography

[1] Lasha Abzianidze, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos. The parallel meaning bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 242–247, 2017.

[2] Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *Proceedings of ICLR-2017 (Volume 1: Long Papers)*, Toulon, France, August 2017.

[3] Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. Polyglot: Distributed word representations for multilingual nlp. In *Proceedings of CoNLL-2013*, pages 183–192, Sofia, Bulgaria, August 2013.

[4] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *Proceedings of the 4th International Conference on Learning Representations (ICLR-2016)*, April 2016.

[5] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *The Journal of Machine Learning Research*, 6:1817–1853, 2005.

[6] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of ACL-2016 (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August 2016.

[7] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of ACL-2015 (Volume 1: Long Papers)*, pages 344–354, Beijing, China, July 2015.

[8] Nicholas Asher. Selectional restrictions, types and categories. *Journal of Applied Logic*, 12(1):75–87, 2014.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR-2015*, 2015.

[10] Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of EMNLP-2015*, pages 349–359, Lisbon, Portugal, September 2015.

[11] Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. Training with exploration improves a greedy stack lstm parser. In *Proceedings of EMNLP-2016*, pages 2005–2010, Austin, Texas, November 2016.

[12] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of EMNLP-2017*, pages 1957–1967, Copenhagen, Denmark, September 2017.

[13] Yonatan Belinkov. *On internal language representations in deep learning: an analysis of machine translation and speech recognition*. PhD thesis, Massachusetts Institute of Technology, 2018.

[14] Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. What do neural machine translation models learn about morphology? In *Proceedings of ACL-2017 (Volume 1: Long Papers)*, pages 861–872, 2017.

[15] Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. On the linguistic representational power of neural machine translation models. *Computational Linguistics*, pages 1–57, 2019.

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

[17] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35 (8):1798–1828, 2013.

[18] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.

[19] Johannes Bjerva, Barbara Plank, and Johan Bos. Semantic tagging with deep residual networks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3531–3541, 2016.

[20] Bernd Bohnet and Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of EMNLP-2012*, pages 1455–1465, Jeju Island, Korea, July 2012.

[21] Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceeding of CoNLL-2006*, pages 149–164, New York, NY, 2006.

[22] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics, 2000.

[23] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP-2014*, pages 740–750, Doha, Qatar, October 2014.

[24] Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. In *Proceedings of EMNLP-2016*, pages 2204–2214, Austin, Texas, November 2016.

[25] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition: a maximum entropy approach using global information. In *Proceedings of CoNLL-2003*, pages 1–7, 2002.

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

[27] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014.

[28] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[29] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[30] Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

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

[32] Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. In *Proceedings ACL-2018 (Volume 1: Long Papers)*, pages 2126–2136, 2018.

[33] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3(Jan):951–991, 2003.

[34] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar): 551–585, 2006.

[35] Leyang Cui and Yue Zhang. Hierarchically-refined label attention network for sequence labeling. In *Proceedings of EMNLP-2019*, pages 4106–4119, 2019.

[36] Hong-Jie Dai, Po-Ting Lai, Yung-Chun Chang, and Richard Tzong-Han Tsai. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *Journal of cheminformatics*, 7(S1):1–10, 2015.

[37] Yann N Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *arXiv preprint arXiv:1502.04390*, 2015.

[38] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC-2006*, pages 449–454, 2006.

[39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[40] Cıcero dos Santos, Victor Guimaraes, RJ Niterói, and Rio de Janeiro. Boosting named entity recognition with neural character embeddings. In *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*, page 25, 2015.

[41] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[42] Finale Doshi-Velez, Mason Kortz, Ryan Budish, Chris Bavitz, Sam Gershman, David O'Brien, Stuart Schieber, James Waldo, David Weinberger, and Alexandra Wood. Accountability of ai under the law: The role of explanation. *arXiv preprint arXiv:1711.01134*, 2017.

[43] Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICLR-2017 (Volume 1: Long Papers)*, Toulon, France, August 2017.

[44] Greg Durrett and Dan Klein. Easy victories and uphill battles in coreference resolution. In *Proceedings of EMNLP-2013*, pages 1971–1982, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[45] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL-2015 (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July 2015.

[46] Jason M Eisner. Three new probabilistic models for dependency parsing: An exploration. In

Proceedings of COLING-1996 (Volume 1), pages 340–345. Association for Computational Linguistics, 1996.

[47] Allyson Ettinger, Ahmed Elgohary, and Philip Resnik. Probing for semantic evidence of composition by means of simple classification tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 134–139, 2016.

[48] Nicolas R Fauceglia, Yiu-Chang Lin, Xuezhe Ma, and Eduard Hovy. Word sense disambiguation via propstore and ontonotes for event mention detection. In *Proceedings of the The 3rd Workshop on EVENTS: Definition, Detection, Coreference, and Representation*, pages 11–15, Denver, Colorado, June 2015.

[49] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of HLT-NAACL-2003*, pages 168–171, 2003.

[50] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[51] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[52] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2016.

[53] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[54] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research*, 3:115–143, 2003.

[55] Rich Caruana Steve Lawrence Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, volume 13, page 402. MIT Press, 2001.

[56] Jesús Giménez and Lluís Màrquez. Svmtool: A general pos tagger generator based on support vector machines. In *In Proceedings of LREC-2004*, 2004.

[57] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedfor-

ward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[58] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE, 1996.

[59] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016.

[60] A Graves. Supervised sequence labelling with recurrent neural networks [ph. d. dissertation]. *Technical University of Munich, Germany*, 2008.

[61] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of ICASSP-2013*, pages 6645–6649. IEEE, 2013.

[62] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[63] Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, et al. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL-2009: Shared Task*, pages 1–18, 2009.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[65] John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of EMNLP-2019*, pages 2733–2743, 2019.

[66] John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of NAACL-2019, Volume 1 (Long and Short Papers)*, pages 4129–4138, 2019.

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

[68] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *Proceedings of ACL-2016*, Berlin, Germany, August 2016.

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

[70] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.

[71] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, 2000.

[72] Jerrold J Katz and Jerry A Fodor. The structure of a semantic theory. *language*, 39(2): 170–210, 1963.

[73] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[75] Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of ACL-2010*, pages 1–11, Uppsala, Sweden, July 2010.

[76] Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of EMNLP-2007*, pages 141–150, Prague, Czech Republic, June 2007.

[77] Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP-2010*, pages 1288–1298, Cambridge, MA, October 2010.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*,

pages 1097–1105, 2012.

[79] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. Distilling an ensemble of greedy dependency parsers into one mst parser. In *Proceedings of EMNLP-2016*, pages 1744–1753, Austin, Texas, November 2016.

[80] Matthieu Labeau, Kevin Löser, Alexandre Allauzen, and Rue John von Neumann. Non-lexical neural architecture for fine-grained pos tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237, 2015.

[81] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-2001*, volume 951, pages 282–289, 2001.

[82] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of NAACL-2016*, San Diego, California, USA, June 2016.

[83] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[84] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[85] Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-rank tensors for scoring dependency structures. In *Proceedings of ACL-2014 (Volume 1: Long Papers)*, pages 1381–1391, Baltimore, Maryland, June 2014.

[86] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[87] Dekang Lin and Xiaoyun Wu. Phrase clustering for discriminative learning. In *Proceedings of ACL-2009*, pages 1030–1038, 2009.

[88] Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/too simple adaptations of

word2vec for syntax problems. In *Proceedings of NAACL-2015*, pages 1299–1304, Denver, Colorado, May–June 2015.

[89] Zachary C Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

[90] Nelson F Liu, Matt Gardner, Yonatan Belinkov, Matthew E Peters, and Noah A Smith. Linguistic knowledge and transferability of contextual representations. In *Proceedings of NAACL-2019, Volume 1 (Long and Short Papers)*, pages 1073–1094, 2019.

[91] Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. Joint entity recognition and disambiguation. In *Proceedings of EMNLP-2015*, pages 879–888, Lisbon, Portugal, September 2015.

[92] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP-2015*, pages 1412–1421, Lisbon, Portugal, September 2015.

[93] Xuezhe Ma and Eduard Hovy. Efficient inner-to-outer greedy algorithm for higher-order labeled dependency parsing. In *Proceedings of the EMNLP-2015*, pages 1322–1328, Lisbon, Portugal, September 2015.

[94] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of ACL-2016*, pages 1064–1074, Berlin, Germany, August 2016.

[95] Xuezhe Ma and Eduard Hovy. Neural probabilistic model for non-projective mst parsing. In *Proceedings of IJCNLP-2017 (Volume 1: Long Papers)*, pages 59–69, Taipei, Taiwan, November 2017.

[96] Xuezhe Ma and Fei Xia. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In *Proceedings of ACL-2014*, pages 1337–1348, Baltimore, Maryland, June 2014.

[97] Xuezhe Ma and Hai Zhao. Fourth-order dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 785–796, Mumbai, India, December 2012.

[98] Xuezhe Ma and Hai Zhao. Probabilistic models for high-order projective dependency

parsing. *Technical Report, arXiv:1502.04174*, 2012.

[99] Xuezhe Ma, Zhengzhong Liu, and Eduard Hovy. Unsupervised ranking model for entity coreference resolution. In *Proceedings of NAACL-2016*, San Diego, California, USA, June 2016.

[100] Xuezhe Ma, Yingkai Gao, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. Dropout with expectation-linear regularization. In *Proceedings of the 5th International Conference on Learning Representations (ICLR-2017)*, Toulon, France, April 2017.

[101] Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-pointer networks for dependency parsing. In *Proceedings of ACL-2018 (Volume 1: Long Papers)*, pages 1403–1414, 2018.

[102] Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. Flowseq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4273–4283, 2019.

[103] Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189. Springer, 2011.

[104] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[105] Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. Dual decomposition with many overlapping components. In *Proceedings of EMNLP-2011*, pages 238–249, Edinburgh, Scotland, UK., July 2011.

[106] Andre Martins, Miguel Almeida, and Noah A. Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of ACL-2013 (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August 2013.

[107] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in transla-

tion: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305, 2017.

[108] Ryan McDonald and Joakim Nivre. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230, 2011.

[109] Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *Proceeding of EACL-2006*, 2006.

[110] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of ACL-2005*, pages 91–98, Ann Arbor, Michigan, USA, June 25-30 2005.

[111] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP-2005*, pages 523–530, Vancouver, Canada, October 2005.

[112] Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. Universal dependency annotation for multilingual parsing. In *Proceedings of ACL-2013*, pages 92–97, Sofia, Bulgaria, August 2013.

[113] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[114] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[115] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[116] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533,

2015.

[117] Vincent Ng. Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of ACL-2010*, pages 1396–1411, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[118] Truc-Vien T. Nguyen, Alessandro Moschitti, and Giuseppe Riccardi. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of EMNLP-2009*, pages 1378–1387, Singapore, August 2009.

[119] Joakim Nivre and Mario Scholz. Deterministic dependency parsing of English text. In *Proceedings of COLING-2004*, pages 64–70, Geneva, Switzerland, August 23-27 2004.

[120] Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June 2007.

[121] Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, et al. Universal dependencies 2.2. 2018.

[122] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

[123] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of ICML-2013*, pages 1310–1318, 2013.

[124] Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon infused phrase embeddings for named entity resolution. In *Proceedings of CoNLL-2014*, pages 78–86, Ann Arbor, Michigan, June 2014.

[125] Nanyun Peng and Mark Dredze. Named entity recognition for chinese social media with jointly trained embeddings. In *Proceedings of EMNLP-2015*, pages 548–554, Lisbon, Portugal, September 2015.

[126] Nanyun Peng and Mark Dredze. Improving named entity recognition for chinese social

media with word segmentation representation learning. In *Proceedings of ACL-2016*, Berlin, Germany, August 2016.

[127] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.

[128] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of EMNLP-2014*, pages 1532–1543, Doha, Qatar, October 2014.

[129] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-2018, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

[130] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. In *Proceedings of LREC-2012*, pages 2089–2096, Istanbul, Turkey, May 2012.

[131] Emily Pitler and Ryan McDonald. A linear-time transition system for crossing interval trees. In *Proceedings of NAACL-2015*, pages 662–671, Denver, Colorado, May–June 2015.

[132] Lawrence Rabiner and B Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[133] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of CoNLL-2009*, pages 147–155, 2009.

[134] Cicero D Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of ICML-2014*, pages 1818–1826, 2014.

[135] Libin Shen, Giorgio Satta, and Aravind Joshi. Guided learning for bidirectional sequence classification. In *Proceedings of ACL-2007*, volume 7, pages 760–767, 2007.

[136] Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural mt learn source syntax? In *Proceedings of EMNLP-2016*, pages 1526–1534, 2016.

[137] David A. Smith and Noah A. Smith. Probabilistic models of nonprojective dependency trees. In *Proceedings of EMNLP-2007*, pages 132–140, Prague, Czech Republic, June

2007.

[138] Anders Søgaard. Semi-supervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 48–52, Portland, Oregon, USA, June 2011.

[139] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[140] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[141] Pontus Stenetorp. Transition-based dependency parsing using recursive neural networks. In *NIPS Workshop on Deep Learning*. Citeseer, 2013.

[142] Xu Sun. Structure regularization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 2402–2410, 2014.

[143] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of CoNLL-2008*, pages 159–177, 2008.

[144] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[145] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings ACL-2015 (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July 2015.

[146] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel Bowman, Dipanjan Das, et al. What do you learn from context? probing for sentence structure in contextualized word representations. In *Proceedings of the 7th International Conference on Learning Representations (ICLR-2019)*, 2019.

[147] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003 - Volume 4*, pages 142–147, Stroudsburg, PA, USA, 2003.

[148] Erik F. Tjong Kim Sang and Jorn Veenstra. Representing text chunks. In *Proceedings of EACL'99*, pages 173–179. Bergen, Norway, 1999.

[149] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of NAACL-HLT-2003, Volume 1*, pages 173–180, 2003.

[150] William Thomas Tutte. *Graph theory*, volume 11. Addison-Wesley Menlo Park, 1984.

[151] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[152] Wenhui Wang and Baobao Chang. Graph-based dependency parsing with bidirectional lstm. In *Proceedings of ACL-2016 (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany, August 2016.

[153] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 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, 2015.

[154] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. In *Proceedings of ACL-2015 (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July 2015.

[155] Yorick Wilks. A preferential, pattern-seeking, semantics for natural language inference. *Artificial intelligence*, 6(1):53–74, 1975.

[156] Jun Xie, Haitao Mi, and Qun Liu. A novel dependency-to-string model for statistical machine translation. In *Proceedings of EMNLP-2011*, pages 216–226, Edinburgh, Scotland, UK., July 2011.

[157] Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. Building a large-scale annotated

chinese corpus. In *Proceedings of COLING-2002*, pages 1–8, 2002.

[158] Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206. Nancy, France, 2003.

[159] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[160] Hao Zhang and Ryan McDonald. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of ACL-2014 (Volume 2: Short Papers)*, pages 656–661, Baltimore, Maryland, June 2014.

[161] Kelly W Zhang and Samuel R Bowman. Language modeling teaches you more syntax than translation does: Lessons learned through auxiliary task analysis. *arXiv preprint arXiv:1809.10040*, 2018.

[162] Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. Dependency parsing as head selection. *arXiv preprint arXiv:1606.01280*, 2016.

[163] Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of EMNLP-2014*, pages 1013–1024, Doha, Qatar, October 2014.

[164] Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June 2011.