Dependency Parsing for Tweets

Yi Zhu

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2017

Committee:

Noah Smith, Chair

Gina-Anne Levow

Program Authorized to Offer Degree:
Department of Linguistics

University of Washington

Abstract

Dependency Parsing for Tweets

Yi Zhu

Chair of the Supervisory Committee:
Dr. Noah Smith
Department of Computer Science & Engineering

This thesis concentrates on the problem of dependency parsing for Twitter texts. Twitter texts, also called tweets, are a typical kind of web domain language with many informal and specific linguistic phenomena (Eisenstein, 2013), which is drawing more attention in NLP research. Although parsing algorithms have achieved huge progress in newswire text data in recent years, it is hard for parsers directly trained on them to achieve comparable results in tweets (Foster et al., 2011a). Therefore, we try to tackle this problem in two aspects, data and model. In the first aspect, we discuss the Twitter specific linguistic phenomena that could cause challenges for creating tweet dependencies, and take them into account within our annotation formalisms. We create a new development set with 210 tweets for the first tweet dependency treebank, Tweebank (Kong et al., 2014). In the second aspect, we propose neural tweet parser, a novel neural dependency parser for tweets. We extend the stack LSTM parser (Dyer et al., 2015) and incorporate character embeddings (Ballesteros et al., 2015) into our word representations. We further explore both out-of-domain data by presenting a cascading model using pre-training and unannotated in-domain data using tri-training to increase the scale of the training data. Experimental results show that our neural tweet parser is over 15 times faster than Tweeboparser (Kong et al., 2014), the previous state-of-the-art parser for tweets. Our parser also benefits from both types of external data, and with tri-training data, our parser outperforms Tweeboparser.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, I am very thankful to my supervisor and thesis advisor Noah Smith, for his guidance and support throughout my whole Master program. I feel very lucky to be able to undertake research under his supervision. He taught me a lot not only in the field of natural language processing (NLP), but also many other aspects. He exemplifies what I should learn in order to become a passionate, mature and insightful researcher just like him. Without his involvement and patience, the thesis would never have been possible.

I am also very thankful to my fantastic thesis reader Gina-Anne Levow. She became my thesis reader at the very last moment during the thesis process and really saved my life. She was very responsible and serious despite the tight timeline and offered me many original comments and feedbacks. Without her generous help and assistance, this piece of texts would never have become a thesis. Beyond the committee, I am grateful to Fei Xia for sharing her invaluable experience on parsing and the annotation of treebank.

I would also like to express my gratitude to my collaborators, Yijia Liu of Harbin Institute of Technology and Lingpeng Kong of Carnegie Mellon University. They have been very generous to help me and answer my silly questions. I benefited a lot from the discussions and work with them. I would like to give my special thanks to Yijia Liu. It is his help and contribution that accelerates the thesis process tremendously.

Many thanks to all of the members and alumni of Noah's Ark and all the other people, who have helped me in the annotation blitz and during the thesis. Your ideas, participation and feedbacks are crucial for this work.

Last but not least, I would like to thank my parents. Their encouragement and faith gave me the strength to overcome all the difficulties and carry on.

Chapter 1

INTRODUCTION

Syntactic parsing is the process of analyzing the syntactic structure of a sentence. It takes a sequence of word tokens as input, and usually outputs a tree structure reflecting the syntactic structure of the input string. It has long been one of the core tasks in natural language processing (NLP) and computational linguistics (CL). Since the thriving of the statistical parsing methods in the early 1990s with the release of the Penn Treebank (PTB) (Marcus et al., 1993), constituent parsing and dependency parsing have been dominating the field with different linguistic perspectives. Dependency parsing, on the basis of the dependency grammar (Tesnière, 1959), is a simple and efficient formalization of syntactic parsing. It has gained a lot of interest recently with more natural representations compared to constituent parsing when dealing with certain linguistic phenomena. For example, the dependency tree structure is usually flatter than the constituent tree structure for the same input sentence, and will therefore have fewer non-projectivity issues, which is especially suited for languages with free word order like Turkish and Czech. Dependency parsing can be useful for many NLP tasks such as information extraction (Nguyen et al., 2009) and machine translation (Katz-Brown et al., 2011; Xie et al., 2011).

In contrast to standardized newswire texts like PTB, web domain and social media texts represent language much closer to our daily life. They are very different from newswire text, containing linguistic challenges including non-standard punctuation, capitalization, spelling, vocabulary, and syntax (Eisenstein, 2013). The study of Twitter (`Twitter.com`) texts, tweets, has become increasingly important in NLP research, because traditional NLP tools trained on newswire texts will simply fail on tweets. Figure 1.1 shows two tweet ex-

```
Perfect ♡ RT @coldplay :  Fix You from the back #ColdplayMinneapolis
http://bit.ly/2dj2WCl

its #awesome u gonna ♡ it Chk out our cooool project on http:// + RT pls
```

Figure 1.1: Tweet Examples

amples. We can see that Twitter has many special tokens such as retweet marker (RT), at-mentions (@coldplay), hashtags (#ColdplayMinneapolis, #awesome), emojis and emoticons (♡), orthographic variations (its, cooool), etc. These tokens never or hardly appear in the newswire texts, and their syntactic functionalities have to be decided by considering context contents, which make parsing challenging for traditional parsers trained only on the newswire texts. Foster et al. (2011a) witnessed a drastic drop of over 10 unlabelled attachment score (UAS), e.g. the percentage of tokens that have the correct head without the arc label, from 90.61 to 73.56 in performance for parsers trained on PTB data, when moving from the in-domain test set to the new Twitter dataset. Foster et al. (2011b) tried various types of parsers on genres of web texts, and pointed out that the parsers had a particular problem with tweets, where conjunctions, adjectival modifiers, adverbial modifiers and clausal complements produced most errors for both dependency and constituent parsers.

Work has been done on dealing with Twitter texts. The first problems addressed on tweets are part-of-speech (POS) tagging and named entity recognition (NER). Gimpel et al. (2011) developed a tagset especially for tweets, manually created a POS tagged dataset, and developed a Twitter POS tagger with features designed for tweets. Owoputi et al. (2013) enlarged the dataset of Gimpel et al. (2011), and incorporated Brown clusters (Brown et al., 1992) obtained from billions of tokens of unlabeled conversational text to further improve the accuracy of the tagger. Ritter et al. (2011) also demonstrated that existing tools for POS tagging, chunking and NER performed quite poorly when applied to tweets, and they built a named entity recognizer for tweets and obtained a 25% increase in F_1 score over

the previous approach for standard texts. Progress has also been made in recent years for parsing tweets. Foster et al. (2011a) annotated a small amount of tweet data of 519 tweets and split them into development and test set. Foster et al. (2011b) considered parsing tweets as a domain adaptation problem, and used the self-training (McClosky et al., 2006a,b) and uptraining (Petrov et al., 2010) methods to improve the parser performance tested on the tweet test data (Foster et al., 2011a). Kong et al. (2014) created a tweet dependency treebank (Tweebank), most of which was built in a day by two dozen annotators with only cursory training in the annotation scheme. Kong et al. (2014) also built a graph-based dependency parser for tweets, Tweeboparser, and achieved decent results on Tweebank test set by leveraging PTB data and Brown clusters.

In this thesis, we are focused on the problem of dependency parsing for tweets. We are primarily interested in tackling challenges in the following aspects:

- Data: How we can create new and high quality tweet dependencies. How we should analyze the Twitter specific linguistic phenomena, and adequately handle them within our annotation formalism.

- Model: How we can design a parser model that can handle annotated tweet data with special annotations inspired by the Twitter specific linguistic phenomena.

1.1 Contributions

The primary contributions of this thesis include:

- We create a new development set for the Twitter dependency treebank, Tweebank (Kong et al., 2014). The new development set has 210 tweets with unlabelled dependencies and has consistent annotations with Tweebank test set.

- We propose a novel neural dependency parser for tweets, neural tweet parser, by extending the stack LSTM parser (Dyer et al., 2015) to allow for Twitter specific linguistic

phenomena. We show that the main model of our neural tweet parser reaches the balance of the tradeoff between accuracy and speed.

- We explore external data to transfer knowledge using PTB data and increase tweet data size using tri-training. We demonstrate the performance of our neural tweet parser can be improved with more external data from both general domain and Twitter specific domain.

1.2 Outline

The remainder of this thesis is organized as follows:

- **Chapter 2** addresses the corresponding Twitter specific linguistic phenomena and creates a new development set for Tweebank.

- **Chapter 3** introduces the neural tweet parser by extending the stack LSTM parser and its variants for the input word representations, and leverages external data to boost the parser performance.

- **Chapter 4** experiments with different models of the neural tweet parser with different training data, and reports the results on Tweebank development and test set compared with Tweeboparser.

- **Chapter 5** summarizes the thesis and directions for future work.

Chapter 2

DATA

In this chapter, we consider the problem of creating new tweet dependencies and address the Twitter specific linguistic phenomena. Following Tweebank annotation conventions and guidelines, we create by ourselves a new development set comprising 210 tweets.

The remainder of this chapter is organized as follows. We first introduce Tweebank and its statistics in Section 2.1. Then we describe the creation of Tweebank development set in Section 2.2. We introduce the general information and statistics of the development set, and the annotation tools we are using. We also discuss the annotation challenges caused by the Twitter specific linguistic phenomena and present our solutions to them.

2.1 Tweebank

Tweebank is a dependency treebank for tweets, most of which was created in a day by two dozen annotators, most of whom had only cursory training in the annotation scheme (Kong et al., 2014). Kong et al. (2014) argue that rapid, small scale annotation efforts performed by imperfectly-trained annotators should be enough to produce a runnable parser given the rapidly changing nature of tweets (Eisenstein, 2013) and thousands of person-hours of annotation labor in general parsing treebanks like PTB. They hope that the disagreement and the errors of annotation could be handled properly by the parser.

Tweebank consists of 918 tweets with 717 in training set and 201 in test set. The tweets are sampled from the Twitter corpus of Owoputi et al. (2013), which are tokenized tweets that contain manually annotated POS tags proposed in Gimpel et al. (2011). The Twitter corpus of Owoputi et al. (2013) consists of OCT27, a corpus with tweets sampled on October 27, 2010 created by Gimpel et al. (2011), and DAILY547, a corpus with one random English

	Train	Test	Dev
Tweets	717	201	210
Unique Tweets	638	201	210
Tokens	9310	2839	3490
Types	3566	1461	1466
Parts	1473	429	463

Table 2.1: Statistics of Tweebank

tweet per day from January 2011 through June 2012 created by Owoputi et al. (2013). Tweebank is drawn roughly equally from both of the corpora (Kong et al., 2014). The statistics of Tweebank are shown in Table 2.1, where types are token types and parts are real sentences or segments within tweets, as one tweet can contain multiple sentences or segments. The training set was annotated in a short time, mostly by less experienced or lower-proficiency annotators and some tweets are annotated by multiple users. The test set was annotated by the most experienced annotators to guarantee that the test set has the highest quality.

2.2 Tweebank Development Set

Similar to the creation of Tweebank, we draw another 210 tweets equally from OCT27 and DAILY547, and manually annotate them. The statistics of the development set can also be seen in Table 2.1. We use the same annotation language and tools as Kong et al. (2014), i.e. Graph Fragment Language (GFL), a text-based notation to annotate parses (Schneider et al., 2013), and a Python Flask web application (Mordowanec et al.). Besides the basic notation of GFL, we also use the special notation of *coordination structures*, but do not explore the underspecification notation as Kong et al. (2014) did, because underspecified structure is not allowed in the final dependency trees.

Chapter 4 will show that our development set is consistent with the test set in annotations.

`bieber is an alien and he went down to earth :O .`

$a** :: {is [went down]} :: {and}

bieber > is < alien < an

he > [went down] < to < earth

Figure 2.1: Example of GFL annotations

2.2.1 Annotation

We follow exactly the same convention and methods Kong et al. (2014) adopted for tweet annotation. We adopt the "Yamada-Matsumoto" (YM) annotation convention (Yamada and Matsumoto, 2003). YM places emphasis on the function words such as auxiliary verbs and prepositions, and makes them the parents of their arguments. For example, the auxiliary verb is the parent of the main verb that conveys the semantic meaning and the preposition is the head word of the prepositional phrase it belongs to.

Apart from the general conventions, Kong et al. (2014) discussed some Twitter specific linguistic phenomena and proposed some methods to tackle the annotation challenges. We discuss them in brief.

- Token Selection: Many tokens in tweets do not have any syntactic function and should be excluded from parsing. These tokens include retweet discourse markers, hashtags, URLs, at-mentioned usernames, punctuation, interjections, emoticons and emojis. They are picked out and annotated attached to a special node (numbered as -3), so that these tokens are not part of the final dependency trees.

- Multiword Expressions: It is difficult to analyze the internal syntactic structure of multiword expressions (MWEs) like proper nouns and lexical idioms (Sag et al., 2002), and very often it is not of great interest for us or downstream tasks. Kong et al. (2014) argue that the annotators should not expend their energy on developing and respecting conventions (or making arbitrary decisions) within syntactically opaque or

idiosyncratic units. Therefore, MWEs are treated as a single node in the tweets, where the annotators do not have to explicitly specify their internal structures. Furthermore, Kong et al. (2014) applied an automatic conversion to the MWE annotations, similar to Johnson (1998), which is integrated into the annotation tools, so that once some phrase is marked as MWE by annotators, its internal structure would be annotated automatically.

- Multiple Roots: There are usually multiple sentences or phrases independent from each other, called "parts", in a single tweet, with or without delimiting punctuation. Kong et al. (2014) allow annotators to decide which span in tweet can be a part, and attach the roots of all parts within a tweet to a final dummy root.

One example of GFL annotations is shown in Figure 2.1. The left side of $>$ is the dependent, and the right side is the head, and for $<$ the rule is reversed. [x] is a MWE notation where "x" inside the square bracket is the MWE. From the example we can see that the tweet has a coordination structure where "$a" is a variable connecting the heads of the two subordinate clauses, "is" and "went down" with the conjunction "and". The annotation follows the YM convention, where "is", as the copula, is the head of the clause. As ":O" is an emoticon, it is not included in the annotation itself. In the CONLL files of Tweebank, labels are usually marked by the placeholder "_" and the MWEs are labelled as "MWE". In coordination structures, the conjuncts are labelled as "CONJ" and conjunctions are labelled as "COORD". These labels are just signs of the different notations we have used in GFL and we do not evaluate them for our results as we only concentrate on unlabelled scores.

The data was annotated by two relatively experienced annotators (including the author of this thesis). They had been trained on a small portion of data and then reached consensus on annotations where there had been initial disagreement. The whole annotation for the development set took around 50 hours for each annotator. The inter-annotator agreement rate on dependencies is around 85%. Major sources of the disagreement are from prepositional

phrase attachment, conjunction, and different understanding of the meaning of the tweets. It should be noted that both annotators are from a non-native English speaking country, just like many of the original Tweebank annotators. This might induce annotation bias caused by annotators speaking languages other than English, and we are willing to explore this effect in future study.

2.3 Summary

In this chapter, we created a new development set for Tweebank. Tweebank development set has 210 tweets and will be proved to have high annotation quality and keeps the annotation consistency with Tweebank test set in the following chapters. Following Kong et al. (2014), we are able to address the Twitter linguistic phenomena easily using GFL as our annotation language.

Chapter 3

A NEURAL DEPENDENCY PARSER FOR TWEETS

In this chapter, we consider the tweet parsing model. We introduce a novel neural dependency parser for tweets, neural tweet parser, by modifying the transition-based stack LSTM parser (Dyer et al., 2015). Furthermore, we leverage other types of data to improve the performance of the parser. We propose a cascading model integrating out-of-domain PTB data, and adopt tri-training to parse massive unannotated tweet data to augment our training data.

The remainder of this chapter is organized as follows. We first review the original stack LSTM parser (Dyer et al., 2015) and its character-based variant (Ballesteros et al., 2015). Then we propose our main model for neural tweet parser by extending the stack LSTM parser. We also present some variants for the input representations of the main model. After that we introduce the cascading model using PTB data and tri-training using unannotated tweet data.

3.1 Stack LSTM Parser

3.1.1 Stack LSTMs

Long Short-Term Memories

Long short-term memories (LSTMs) are a special kind of recurrent neural network (RNN) for ameliorating the vanishing gradient problem when training long sequences in RNNs (Hochreiter and Schmidhuber, 1997; Graves, 2013). At each time step t, LSTMs read in the input vector $\boldsymbol{x}_t$, and process it with three gates. These gates control and protect each LSTM cell, deciding how much of the current input can pass into the memory cell ($\boldsymbol{i}_t$) and how much

of the previous memory cell to "forget" ($\boldsymbol{f}_t$). An extra cell state ($\boldsymbol{c_t}$) and a hidden state $\boldsymbol{h}_t$ is also calculated by LSTMs. The LSTMs Dyer et al. (2015) use are another variant of the standard LSTMs, introduced by Gers et al. (2003), where each gate will look at the previous cell states during the calculation and update, which is called "peephole connections". Each time, $\boldsymbol{h}_t$ is updated as follows:

$$
\begin{aligned}
\boldsymbol{i}_t &= \delta(\mathbf{W}_{ix}\boldsymbol{x}_t + \mathbf{W}_{ih}\boldsymbol{h}_{t-1} + \mathbf{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \delta(\mathbf{W}_{fx}\boldsymbol{x}_t + \mathbf{W}_{fh}\boldsymbol{h}_{t-1} + \mathbf{W}_{fc}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\mathbf{W}_{cx}\boldsymbol{x}_t + \mathbf{W}_{ch}\boldsymbol{h}_{t-1} + +\boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \delta(\mathbf{W}_{ox}\boldsymbol{x}_t + \mathbf{W}_{oh}\boldsymbol{h}_{t-1} + \mathbf{W}_{oc}\boldsymbol{c}_t + \boldsymbol{b}_o) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)
\end{aligned}
$$

where δ is the element-wise sigmoid function, *tanh* is the hyperbolic tangent function, and $\odot$ is the element-wise product (Hadamard product). A third gate $\boldsymbol{o}_t$ is used to add further nonlinearity.

Stack Long Short-Term Memories

Conventional LSTMs are a sequence model in a left-to-right order. Stack LSTMs in Dyer et al. (2015) augment the LSTMs with a stack pointer that can track the last LSTM cell currently along with its states including the cell states and hidden states to determine which cell to use when computing new LSTM cells. In addition, stack LSTM has a *push* and *pop* operation functioning just like a stack. The *push* operation adds a new LSTM cell after the stack pointer and moves the stack pointer to the newly added cell, and the *pop* operation reverts the pointer to the previous element, as shown in Figure 3.1.

3.1.2 Dependency Parser

The Stack LSTM parser preserves the framework of the transition-based parser, i.e. a buffer (B) of words to be processed and a stack (S) of partially constructed dependency subtrees,

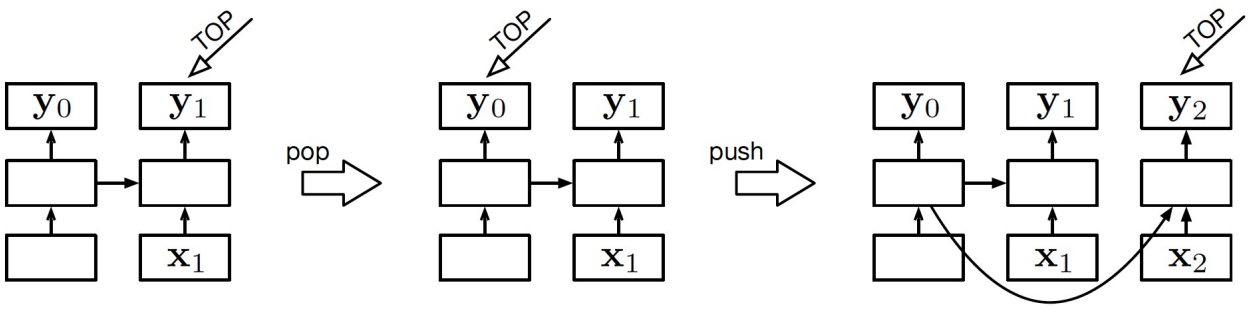

Figure 3.1: Stack LSTM with three configurations: a stack with a single element (left), the result of a *pop* operation to this (middle), and then the result of applying a *push* operation (right) (Dyer et al., 2015).

seen in Figure 3.2. In addition, an action history stack (A) is added to record the actions taken by the parser. Each of the three components is represented by a stack LSTM and therefore three continuous-space vectors will be generated as the representations denoted as $\boldsymbol{b}_t$, $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ at time step t. A hidden layer is added to represent the whole parser state, defined as follows:

$$\boldsymbol{p}_t = \max\{0, \mathbf{W}[\boldsymbol{s}_t; \boldsymbol{b}_t; \boldsymbol{a}_t] + \boldsymbol{b}\} \tag{3.1}$$

where ";" is the concatenation operation for two vectors.

Then $\boldsymbol{p}_t$ is used to compute the probability of the parser actions using softmax function (output layer):

$$p(z_t \mid \boldsymbol{p}_t) = \frac{\exp\left(g_{z_t}^T \boldsymbol{p}_t + q_{z_t}\right)}{\sum_{z' \in \mathcal{A}(S,B)} \exp\left(g_{z'_t}^T \boldsymbol{p}_t + q_{z'_t}\right)} \tag{3.2}$$

where the set $\mathcal{A}(S, B)$ represents the valid actions that may be taken given the current contents of the stack and buffer.

Finally, the conditional probability is calculated assuming that each action is independent, as shown in Equation 3.3.

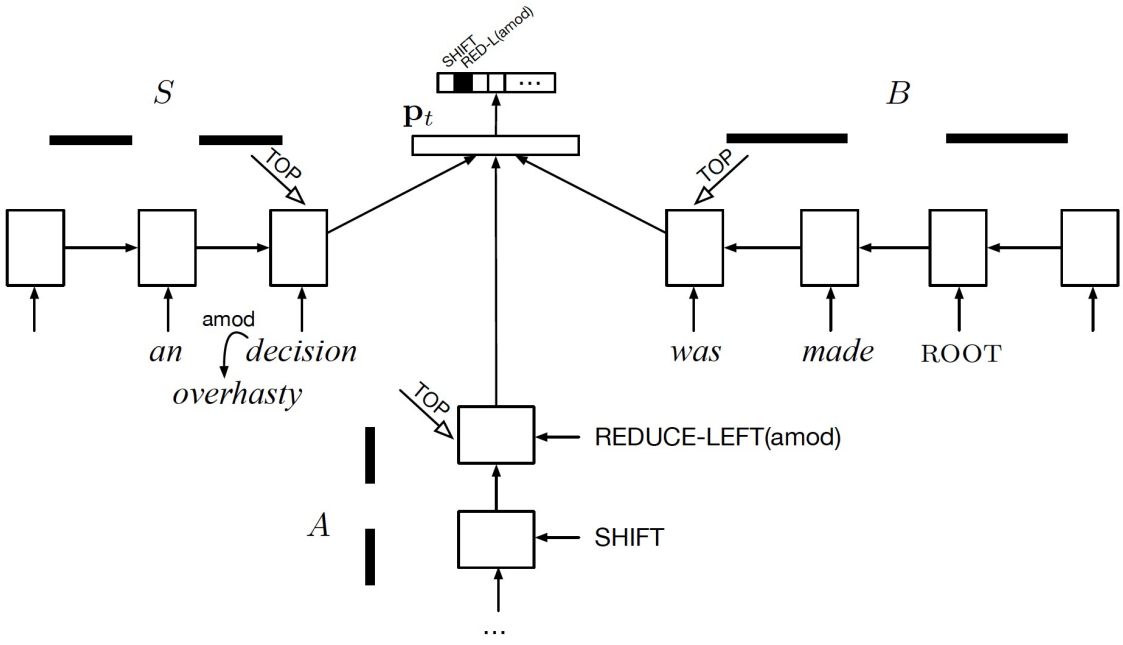

Figure 3.2: Parser state while parsing the sentence "an overhasty decision was made". S denotes the stack of partially constructed dependency subtrees; B is the buffer of words to be parsed; and A is the stack of the history of actions taken by the parser (Dyer et al., 2015).

$$p(\boldsymbol{z} \mid \boldsymbol{w}) = \sum_{t=1}^{|\boldsymbol{z}|} p(z_t \mid \boldsymbol{p}_t) \tag{3.3}$$

where $\boldsymbol{w}$ is the sequence of the words representing the sentence, and $\boldsymbol{z}$ is any valid sequence for parsing the sentence.

During the training, $\boldsymbol{z}$ would be the gold standard transition sequence, and $p(\boldsymbol{z} \mid \boldsymbol{w})$, i.e. the conditional probability of the gold standard parse given sentence, is maximized. During the testing, the parser greedily selects the action with the highest probability for every parse state, and we will eventually obtain an action sequence generated by the parser for every input sentence.

$$\textsc{LeftArc (LA)} \quad \frac{([\sigma \mid \ s_1, s_0], \beta, A)}{([\sigma \mid \ s_0], \beta, A \cup \{s_1 \leftarrow s_0\})}$$

$$\textsc{RightArc (RA)} \quad \frac{([\sigma \mid \ s_1, s_0], \beta, A)}{([\sigma \mid \ s_1], \beta, A \cup \{s_1 \rightarrow s_0\})}$$

$$\textsc{Shift (SH)} \quad \frac{(\sigma, [b \mid \beta], A)}{([\sigma \mid \ b], \beta, A)}$$

Figure 3.3: Arc-standard transition system used in the stack LSTM parser

Transition System

The parser is based on the arc-standard transition system (Nivre, 2004) as shown in Figure 3.3. There are three actions in total, LEFTARC, RIGHTARC and SHIFT. The SHIFT action takes the token in the head of B, and pushes it into S. The LEFTARC attaches the second token in S to the first token, and then pops the second token from S. The RIGHTARC is similar with LEFTARC, except that the head word would be the second token in S.

Whenever a LEFTARC or RIGHTARC action is taken, a composition function will be triggered to compose the head ($\boldsymbol{h}$), dependent ($\boldsymbol{d}$) and action, relation tuple ($\boldsymbol{r}$) to a single vector $\boldsymbol{c}$, representing a composed subtree, seen in Equation 3.4. It is applied in a hierarchically recursive way so that there will be a final vector in the end representing the dependency tree in the stack. The input vectors are either the word representations of tokens (Section 3.2.1) or a subtree representation $\boldsymbol{c}$. We should note that $\boldsymbol{r}$ represents a tuple of the action and its relation. For example, in the tweet parsing scenario, as discussed in Section 2.2.1, the labels can be "_", "CONJ", "COORD" and "MWE". $\boldsymbol{r}$ is thus from all the possible and valid combinations of actions and relations.

$$\boldsymbol{c} = tanh(\mathbf{U}[\boldsymbol{h}; \boldsymbol{d}; \boldsymbol{r}] + \boldsymbol{e}) \tag{3.4}$$

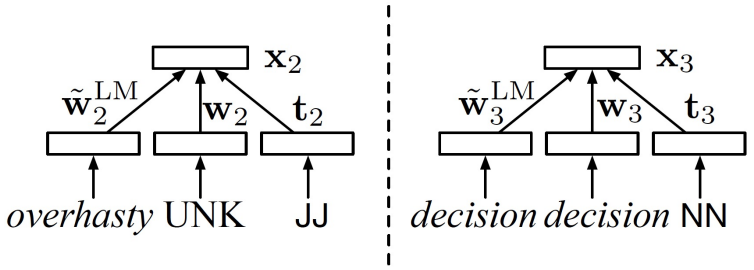

Figure 3.4: Word representations of Dyer et al. (2015)

3.1.3 Word Representations

Different word representations for the input of B have been proposed by Dyer et al. (2015) and Ballesteros et al. (2015). Dyer et al. (2015) use the word type vector ($\boldsymbol{w}$), a fixed vector representation from a neural language model ($\widetilde{\boldsymbol{w}}_{LM}$) and a learned representation ($\boldsymbol{t}$) of the POS tag of the token, provided as auxiliary input to the parser, shown in Equation 3.5 and Figure 3.4. The advantage of this method is that when an out-of-vocabulary (OOV) word comes, we can look it up in the language model representations to mitigate the effect of "UNK".

$$\boldsymbol{x} = \max\{0, \mathbf{V}[\boldsymbol{w}; \widetilde{\boldsymbol{w}}_{LM}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3.5}$$

Alternatively, Ballesteros et al. (2015) use bi-directional character-based embeddings instead of word type embeddings to handle OOVs and capture the morphological features among words, shown in Equation 3.6 and Figure 3.5. The advantage of this method is that as we only have to encode characters instead of words, the number of parameters will be reduced dramatically. It is also robust to the orthographical variations, which is extremely suitable for tweet case.

$$\boldsymbol{x} = \max\{0, \mathbf{V}[\overrightarrow{\boldsymbol{w}}; \overleftarrow{\boldsymbol{w}}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3.6}$$

In practice, one can use either Equation 3.5 or 3.6 as input word representations, or

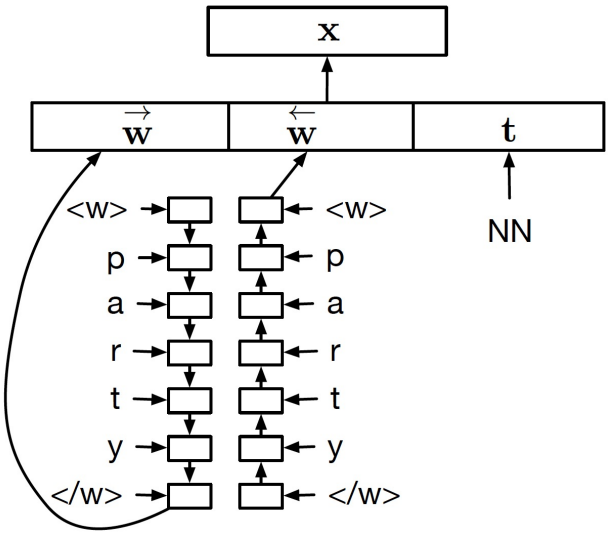

Figure 3.5: Word representations of Ballesteros et al. (2015)

even combine both through concatenation. We will discuss the combinations of different components in Section 3.2.1.

3.2 Neural Tweet Parser

3.2.1 Main Model

Our main model of the neural tweet parser follows the same framework described in Section 3.1.2, seen in Figure 3.6. We make several modifications to the transition system and the word representations. We discuss them in detail.

Adding the `SKIP` *Action*

As is discussed in the previous chapter, we have encountered different types of annotation challenges of tweets, of which the biggest for us is the problem of token selection. Kong et al. (2014) developed a first-order sequence model of the token selector to identify unselected tokens (discourse retweet markers, at-mentioned usernames, etc.) with 97.4% average

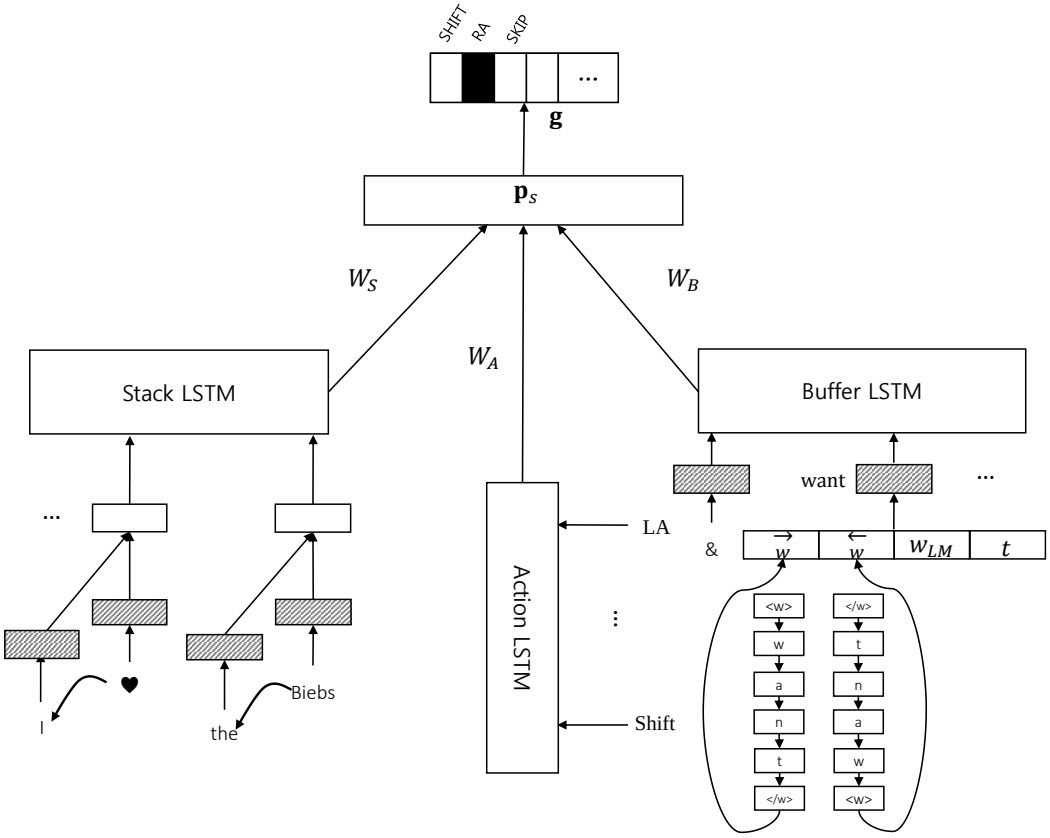

Figure 3.6: Network architecture of neural tweet parser

accuracy using the structured perceptron (Collins, 2002). After eliminating unselected tokens, they use the remaining sentence as input for their parser. We think that instead of using a pipeline model, it could be solved more compactly by integrating the module into our parser. Our idea is simple, that as long as we discard unselected tokens before SHIFTing them to the stack, they would not be included in the syntactic analysis anyway. Therefore, we change our arc-standard system by adding a new operation called SKIP, which simply dumps a token from the head of the buffer. Our new transition system is shown in Figure 3.7 and the new action SKIP is added to the action set in Figure 3.6.

$$\textsc{LeftArc (LA)} \quad \frac{([\sigma \mid \ s_1, s_0], \beta, A)}{([\sigma \mid \ s_0], \beta, A \cup \{s_1 \leftarrow s_0\})}$$

$$\textsc{RightArc (RA)} \quad \frac{([\sigma \mid \ s_1, s_0], \beta, A)}{([\sigma \mid \ s_1], \beta, A \cup \{s_1 \rightarrow s_0\})}$$

$$\textsc{Shift (SH)} \quad \frac{(\sigma, [b \mid \beta], A)}{([\sigma \mid \ b], \beta, A)}$$

$$\textsc{Skip (SK)} \quad \frac{(\sigma, [b \mid \beta], A)}{(\sigma, \beta, A)}$$

Figure 3.7: Transition system of neural tweet parser

Word Representations

We decide to use the word representations of Ballesteros et al. (2015) plus an extra entry of fixed neural language model representations described in Dyer et al. (2015) for our main model, as shown in Equation 3.7 and Figure 3.6. We hope this combination could capture most lexical phenomena of tweets and reach a good tradeoff between speed and accuracy.

$$\boldsymbol{x} = \max\{0, \mathbf{V}[\overrightarrow{\boldsymbol{w}}; \overleftarrow{\boldsymbol{w}}; \widetilde{\boldsymbol{w}}_{LM}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3.7}$$

In our experiments, we also compare the main model with the ones incorporating more features to see if they could help improve the parsing performance. The first feature is the learnable word embeddings $\boldsymbol{w}$. After adding it to our word representations, the new representations become just the union of the two methods described in Ballesteros et al. (2015) and Dyer et al. (2015), shown in Equation 3.8.

$$\boldsymbol{x} = \max\{0, \mathbf{V}[\overrightarrow{\boldsymbol{w}}; \overleftarrow{\boldsymbol{w}}; \boldsymbol{w}; \widetilde{\boldsymbol{w}}_{LM}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3.8}$$

Brown clusters (Brown et al., 1992) are found to also be useful in parsing (Koo et al., 2008) and POS tagging (Owoputi et al., 2013). Kong et al. (2014) use 4, 6 and full bit cluster representations as features for their arc-factored model. We also add them to the previous

input word representations as defined in Equation 3.9 without the learnable word embeddings and Equation 3.10 with learnable word embeddings.

$$\boldsymbol{x} = \max\{0, \boldsymbol{V}[\boldsymbol{br}_4; \boldsymbol{br}_6; \boldsymbol{br}_f; \overrightarrow{\boldsymbol{w}}; \overleftarrow{\boldsymbol{w}}; \widetilde{\boldsymbol{w}}_{LM}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3.9}$$

$$\boldsymbol{x} = \max\{0, \boldsymbol{V}[\boldsymbol{br}_4; \boldsymbol{br}_6; \boldsymbol{br}_f; \overrightarrow{\boldsymbol{w}}; \overleftarrow{\boldsymbol{w}}; \boldsymbol{w}; \widetilde{\boldsymbol{w}}_{LM}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3.10}$$

where the subscript represents the digit of the cluster bit strings. The embeddings of the Brown clusters are updated during the training stage.

3.2.2 Cascading Model

Although we expect that tweets have different syntactic characteristics from data of domains like newswire, it is really hard to ignore such a large amount of high-quality syntactic data, especially for PTB. In effect, there are still a lot of invariable linguistic properties between tweets and PTB and we could learn such knowledge from PTB.

Kong et al. (2014) use a simple, stacking-inspired incorporation of PTB information into their model. During the decoding, they run a state-of-the-art parser trained on PTB to parse the input tweet. The parser will produce scores of possible arcs between tokens in the tweet. They add such scores as features into the new parser trained on Tweebank, and run the new parser to parse the input tweet again. Although it proved to be very useful, the real speed is actually at least two times slower than the parser without PTB features as it is parsing the original tweet twice.

However, it is very easy for a neural network to incorporate features from different models or data. Our method is inspired by Guo et al. (2016), where they share parameters of the stack LSTM parsers for different languages. We therefore propose a cascading model for our neural tweet parser. We first pre-train a stack LSTM parser on PTB data, and share all of its parameters with our neural tweet parser. We initialize our neural tweet parser with corresponding parameters in the PTB pre-trained parser, and then further fine-tune them

on Tweebank.

Depending on the different word representations we use, we maintain a character set, a vocabulary set, a Brown cluster set and an action set, which contain the union of elements in both PTB and Tweebank, so that the model can share the same parameters in both PTB and Tweebank datasets.

We hope that given the limited training data of Tweebank, the cascading model will provide the parser with a good initialization point nearer to better local minima so that the parser can converge more quickly to avoid overfitting and reach a better result.

3.2.3 Tri-training

There are only around 1000 annotated tweets in Tweebank, while an abundance of unannotated tweets are available to explore. Therefore, it is important to make use of the data of such large scale so that we can have more training data of the tweet domain, which could be arguably more useful than out-of-domain PTB data. One potential way to acquire additional training data is to parse the unannotated data with multiple parsers that have been previously trained. Self-training re-trains a single model iteratively in certain settings (McClosky et al., 2006a,b), and Petrov et al. (2010) use a slow but accurate parser for uptraining a faster but less accurate parser.

As we already have an off-the-shelf parser for tweets, Tweeboparser (Kong et al., 2014), it is easy to adopt the tri-training (Zhou and Li, 2005) used in parsing (Søgaard and Rishøj, 2010; Li et al., 2014; Weiss et al., 2015). We use our trained neural tweet parser and Tweeboparser to parse a large amount of tweet data. Then we pick the tweets whose parse trees are the same produced by two parsers. We use these tweets along with their parse trees as additional training data to further train our neural tweet parser. The intuition behind this idea is that the confidence that the parse tree is the right one is much higher given two generated parse trees are the same, because two parsers are biased differently, and reaching a consensus suggests that the results are probably correct. We show in the next chapter that tri-training can improve our neural tweet parser effectively.

3.3 Summary

In this chapter, we have presented the neural tweet parser, a neural network dependency parser for tweets, by extending the stack LSTM parser. We introduced the architecture of the stack LSTM parser and its character-based variant for input word representations. Based on the stack LSTM parser, we described the extensions of our main model for the neural tweet parser by adding SKIP action and adopting the bi-directional character embeddings, POS embeddings and pre-trained language model embeddings as input word representations. We also discussed some variants of the input word representations for our main model. Then we introduced the cascading model by pre-training on PTB data, and use tri-training for data augmentation.

Chapter 4

EXPERIMENTS AND RESULTS

In this chapter, we perform different experiments on the neural tweet parser and report the corresponding results with some discussions. We find that our parser is about 15 times faster than the previous state-of-the-art parser for tweets (Kong et al., 2014), Tweeboparser, and also perform better evaluated on UAS F-scores, when trained on extra training data obtained from tri-training.

The remainder of this chapter is organized as follows. We first describe the content of the experiments, their settings and different configurations of our neural tweet parser. Then we report the result of each experiment, and analyze them quantitatively and qualitatively.

4.1 Experiments

4.1.1 Experimental Setup

We perform three experiments on Tweebank, PTB and unannotated tweet corpus, using different parser configurations, and compare our results with Kong et al. (2014). We train the parser with different training data, depending on the task, tune the hyperparameters on the new development set, and test the parser on the test set. The POS tags used as input for the parser are gold standard in the training data, predicted by a first-order maximum entropy Markov POS tagger with features specifically for tweets and online texts (Owoputi et al., 2013) in both development set and test set, which is also the default settings in Kong et al. (2014). During training, we convert non-projective dependency trees to pseudo-projective dependency trees, as our parsers still cannot handle non-projective trees in general. But we will not do any conversion for the development and test set.

Experiments on Different Word Representation Configurations

We experiment with different word representation configurations to the parser, as described in 3.2.1, and compare them in accuracy and speed. According to the configurations, we switch off or on the corresponding input entries. To add Brown clusters to our dataset, we attach different Brown cluster bits to the end of the columns in Tweebank.[1]

Experiments on Cascading Model

The experiment is to use the cascading model, which is pre-trained on PTB data with different epochs, to test if PTB data could really provide extra knowledge and boost the parsing performance. PTB data used for cascading model needs to be preprocessed so as to keep the conformity to Tweebank, described as follows:

1. We use the PennConverter tool with head rule options selected to approximate Tweebank annotation conventions.[2]

2. We convert the POS tags of PTB to tweet POS tags proposed in Gimpel et al. (2011) according to the mapping table.

3. We use the default relation ("_") of Tweebank to replace all the relations of PTB. This might create a gap between the relations of PTB and those of Tweebank, and we leave this open for the future work.

We first analyze the optimal pre-training epoch for the cascading model. We pre-train our main model with different epochs, and save the parameters for these epochs. Then we initialize another parser with these parameters, and continue to train it using Tweebank training data, and test on the development set and test set. The parsers in two stages do not have to have the same configuration, as the saved parameter set already includes all

[1]Brown clusters are obtained from `http://www.cs.cmu.edu/~ark/TweetNLP/clusters/50mpaths2`

[2]`http://nlp.cs.lth.se/software/treebank_converter`

the parameters needed for any configuration. Once we find the optimal epoch, we use that parameter set as the initialization, and repeat the second experiment.

Experiments on Tri-training

The third experiment is to use the massive amount of unlabelled tweet corpus and tri-training to further improve the parser. We adopt the following steps:

1. We obtain the unlabelled tweets from 1% of the Twitter streams in July 2016 from ArchiveTeam.[3] We pick out all the English tweets using the *lang* attribute in the stream and *langid.py*.[4] It results in over 40 million unlabelled English tweets.

2. We use our main model, the one using the character-based architecture, and Tweeboparser to parse these tweets, which produce over 8 million tweets, where their parse trees are identical for both parsers. We further discard tweets whose length are less than 10, and get around 3.7 million tweets in total. Then we sample different sizes of instances from them, and add them to the new training data.

3. We use the tri-training data, Tweebank training set and PTB data to train the main model and the cascading model, and compare them with the original main model and Tweeboparser on test set.

As the number of the new annotated tweets from tri-training is much larger than Tweebank training set, it could cause severe bias if we directly train the parser on the mixed data. We decide to train them separately. We train only on one dataset in every mini batch, whose size has the same proportion on both datasets, so that Tweebank can guide the parser roughly as equally as tri-training data. For cascading model, we directly use the optimal parameter set learned in the previous experiments. Although the augmentation of the training data might

[3] `https://archive.org`

[4] `https://github.com/saffsd/langid.py`

require different optimal parameters, we found that the previous optimal parameter set can already give the parser a good initialization point, and we leave this problem as future work.

4.1.2 Settings

We use the deep learning C++ library Dynet (Neubig et al., 2017) to implement the neural tweet parser. We tried different hyperparameters in stack LSTM parser on the development set, and found that the settings of Dyer et al. (2015) give us pretty decent results. Therefore, we decide to keep all the hyperparameters the same as in Dyer et al. (2015). We use Twitter GloVe (Pennington et al., 2014) as our pre-trained language model embeddings, acquired on 2B tweets using the GloVe algorithm. It is good at capturing the semantic relations between words. We experimented with all the available dimensions of Twitter GloVe from 25 to 200. We found that the parsing performance generally increased with more dimensions, and 100 and 200 dimensions gave us only marginal difference on the results, so we decide to use 100 as the pre-trained embedding dimension. We use weight decay in Dynet as a way of regularization.

We restrict total training epoch to a maximum of 15, because the training accuracy usually converged before that epoch. The mini batch in training is 100 for both Tweebank and PTB, and have the same proportion for tri-training data as Tweebank accordingly, described previously.

We basically follow the parameter optimization of Dyer et al. (2015), where we use stochastic gradient descent with an initial learning $\eta_0 = 0.1$ and the learning rate was updated on each pass through the training data as $\eta_t = \eta_0/(1 + \rho t)$, with $\rho = 0.1$ and where t is the number of epochs completed. We also add momentum (Sutskever et al., 2013) with the default value $\mu = 0.9$ in Dynet as we found that momentum leads to stable trajectory and faster convergence in our experiments.

We **only** use our main model for both training and testing in tri-training and the pre-training stage of the cascading model. We should note that some of the parameters are only initialized randomly and not trained during pre-training, as it is difficult to disentangle the

saved model parameters in Dynet. We hope that we could solve this technical problem in the near future.

For the cascading model, the epoch ranges from 0 to 10 with step size 1. For tri-training, we sample 5k, 10k, 20k and 50k instances from the final tweet data after tri-training as additional training data.

The evaluation metric is the same used in Kong et al. (2014), which is an F-score of UAS, where the token selection process contributes to the potentially unequal number of tokens in the final dependency trees. In addition to this, punctuations are also regarded as unselected tokens and we do not evaluate them during testing.

As the random initialization of corresponding parameters affects the results a lot, during testing, depending on the tasks, we run the parser several times with random initialization and report the average score of the these runs.

4.1.3 Configurations

Different configuration notations are denoted as follows:

- **main**: Main model described in Section 3.2.1, using character-based word representations described in Equation 3.7.
- **main + word**: Main model plus learnable word embeddings described in Equation 3.8.
- **main + brown**: Main model plus Brown clusters described in Equation 3.9.
- **main + word + brown**: Main model plus learnable word embeddings and Brown clusters described in Equation 3.10.
- **cas**: Cascading model described in Section 3.2.2. It must combine some model of the parser. For example, **main + cas** means the main model is initialized by the pre-trained parameters.

- **tri**: Tri-training described in Section 3.2.3.

- **tweebo**: The main parser result in Kong et al. (2014) using Brown clusters and PTB features. We also have **brown** and **PTB** options for ablation test. For example, **tweebo - PTB** represents the main parser of Kong et al. (2014) without PTB features.

4.2 Results

We show our results of different experiments. We report the best UAS F-scores in the development set and the scores in the test set using the same set of parameters that achieved the best scores in the development set.

From Table 4.1 we can see the results for the experiments on different input word representation configurations in Section 4.1.1. The results are averaged results of 3 runs. The average standard deviation of the four configurations is 0.002 on the development set and 0.012 on the test set. It indicates that despite the scores on the test set are more volatile, the range of the variability is still rather small. We found that in all cases, there is clearly a score gap between the two datasets, the scores of the development set being lower than those of the test set. We think the main reason is the intrinsic data discrepancy between the development set and the test set that caused such gap, and the development set is considered "harder" than the test set. One possible evidence is that all the tweets in the development set came from the incomplete annotations during the creation of Tweebank, which implies that these tweets are generally more difficult to annotate. For different configurations, with introducing more parameters, the scores increase only a little, and for *main + word* the scores even drop about 2 points. Therefore, it is not that clear how much gain we get by incorporating learnable word embeddings and Brown cluster embeddings. It is probable that most of the lexical information is already captured by the character-based representations, and the other two embeddings only contribute very limitedly. On the other hand, more parameters certainly cause the decoding speed to decrease significantly, dropping from 305 tweets/s with our main parser configuration to 173 tweets/s with both word and Brown clus-

	Dev set	Test set	Decoding Speed (tweets/s)
main	76.4	77.1	**305**
main + word	75.3	75.6	190
main + brown	76.5	77.3	188
main + word + brown	**76.8**	**77.7**	173

Table 4.1: Results of different word representation configurations

ter parameters. We argue that our main parser reaches the balance between the accuracy and speed, and it is further supported in the experiment results of cascading model.

Then we report the results for the experiments on cascading model in Section 4.1.1.

Figure 4.1 shows the effects of the pre-training epoch on the parsing results. The results are only one run based on the same fixed random seed used in the pre-training stage, so that the only variable is the initialized parameters pre-trained with different epochs. Both scores in development and test set are fluctuating, but it is clear that from epoch 0 to 3, both scores are increasing, whereas from epoch 4 there is no more clear trend. Nevertheless, the scores in the development set after epoch 3 have never surpassed that in epoch 3. It may be due to the fact that in the beginning, our parser is learning the knowledge of PTB data, which is common in English and also useful for parsing tweets. When we continue the pre-training, the parser gruadually overfits PTB data, learning more specific knowledge about PTB structure and style, some of which is not helpful and could harm the parser's performance on Tweebank. We therefore select epoch 3 as the optimal epoch in terms of training time and accuracy, and repeat the second experiment again, to compare the cascading model with different configurations.

The settings of Table 4.2 are almost the same as Table 4.1, except that the scores, again averaging 3 runs, are from the cascading model. We can easily see that by pre-training the parameters and injecting PTB knowledge, the cascading model does help improve the parser performance consistently in all the configurations. The average score improvement is 3.4

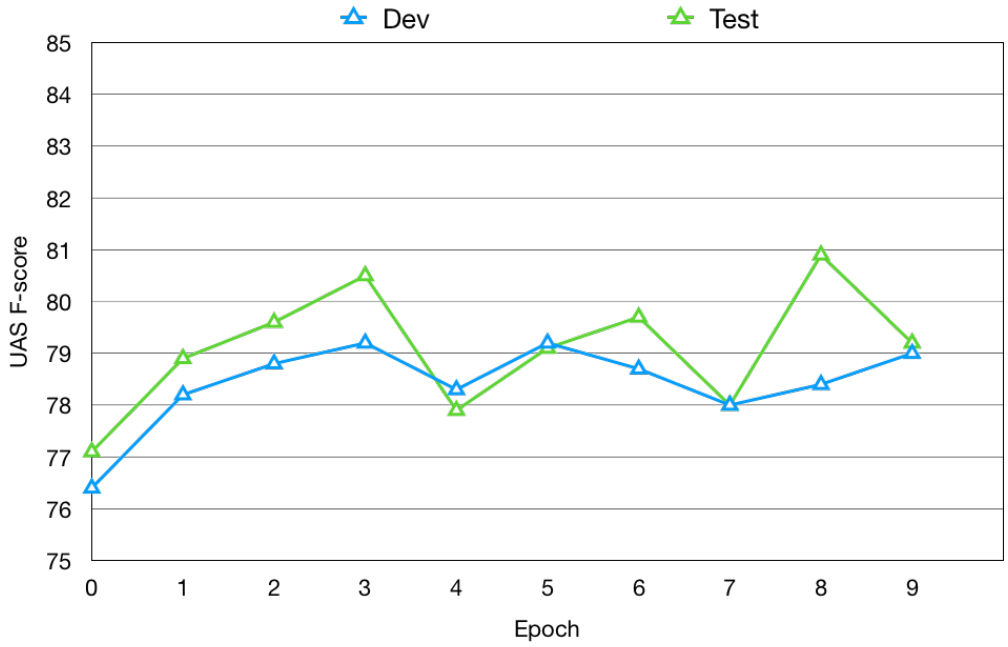

Figure 4.1: Relations between the pre-training epoch and the UAS F-score

in the development set and 3.0 in the test set, and the biggest improvement is 4.4 and 3.8 respectively. Secondly, from the previous statistics of the score improvement, despite the score gap between two datasets, it is bridged by 0.4 averagely, and the score change in the development set can indicate the change in the test set very well. It supports our claim that our annotations of the development set are of very high quality, and consistent with the annotation guidelines adopted in the test set. Besides, we conjecture that there might be more general linguistic structures that can be learned from PTB data in the development set, so the score improvement on the development set is larger than the training set, although the improvement is not salient. At last, it should be noticed that the test score difference among configurations is also smaller, especially when comparing **main + word + brown + cas** with other configurations. It suggests that most extra information Brown clusters provide

	Dev set	Test set
main + cas	79.5	80.1
main + word + cas	**79.7**	79.4
main + brown + cas	79.6	**80.2**
main + word + brown + cas	**79.7**	80.0

Table 4.2: Results of cascading model with different word representation configurations

has already been somehow learned during pre-training. It could be attributed to the fact that POS embeddings have acquired this information from PTB data, and they can replace the role of Brown clusters. We show this to also support our choice of the character-based architecture as the input word representations of our main model.

In experiments on tri-training in Section 4.1.1, Figure 4.2 shows that tri-training can effectively boost both main model and cascading model, giving them around 2 to 4 point gain. It implies that **Tweebank training set** is simply not enough in quality and quantity for parser to learn the general and Twitter specific linguistic structures, as there are only around 600 unique tweets and most of them were annotated by annotators with only cursory training. Using tri-training, gains can be achieved by adding only automatically generated training data, which leads to the strong and consistent supervision on both linguistic structures that the parser can learn. If we regard the cascading model without tri-training data as having encoded the information on general linguistic structures, we could draw previous conclusion based on the improvement on both models, and the contribution of the information from both structures seems roughly equal, observed from general + Twitter specific structures for around 4 point gain for main model versus only Twitter specific structure for around 2 point gain for cascading model. Another interesting point is that the optimal data size in tri-training for both model could be different, as the main model achieves the best with 10k tweets and 20k for the cascading model respectively. It is a little surprising that the best results are not achieved by using the most data, 50k, in the development set of the

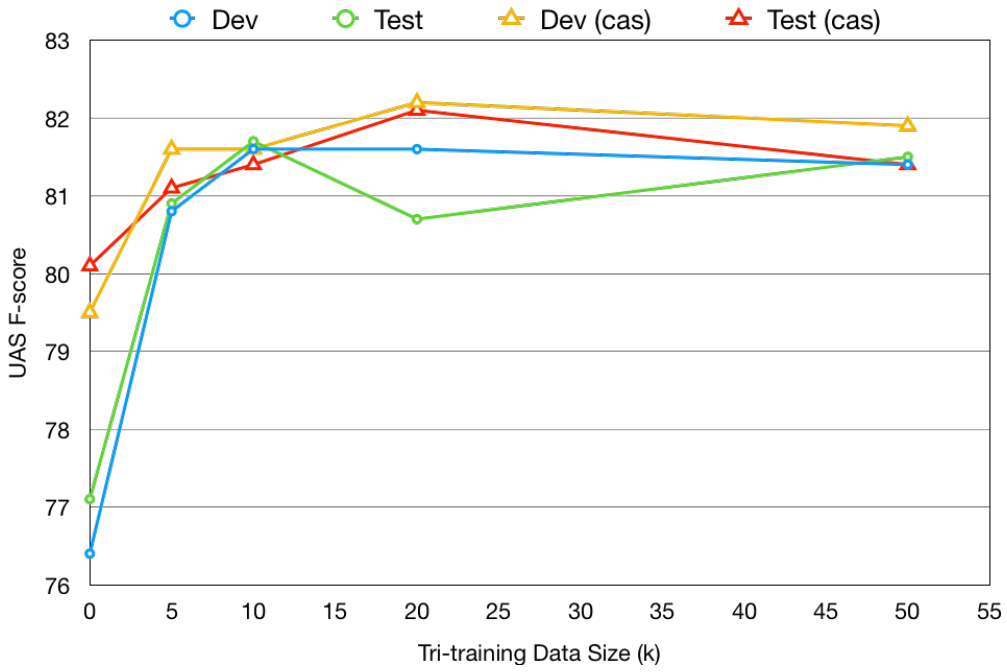

Figure 4.2: Relations between the size of tri-training data and the UAS F-score

main model and the cascading model. For the main model, we think that as the score is averaged over only 3 runs and our development and test set are rather small, it is possible that fluctuations exist. Actually, the scores for our main model with 50k are almost the same as those in 10k. For the cascading model, as the optimal pre-training epoch we use here is tuned when only using Tweebank training set as training data, it may not be the same case for tri-training data. We did not experiment with enough sample points to draw firm conclusions. These issues need to be investigated in detail.

Finally, we compare our neural tweet parser with Tweeboparser in corresponding configurations. We select the configurations of our parser most close to those of Tweeboparser and report the scores on the test set. Seen from Table 4.3, our main model falls behind the result of the counterpart of Tweeboparser (*tweebo - brown - PTB*) by 2.4 points. Leveraging PTB

	Test set	Decoding Speed (tweets/s)
main	77.1	**305**
main + brown	77.3	188
main + cas	80.1	-*
main + brown + cas	**80.2**	-
main + tri	81.7	-
main + cas + tri	**82.1**	-
tweebo - brown - PTB	79.5	21.6
tweebo - PTB	80.2	20.5
tweebo - brown	**81.2**	21.3
tweebo	80.9	20.3

* The decoding speed of both cascading model and the model and that with tri-training data is considered the same as their original models.

Table 4.3: Comparing neural tweet parser with Tweeboparser (Kong et al., 2014)

data helps both our parser and Tweeboparser. *main + cas* improves 3 points with respect to our main model, and *tweebo - brown* improve 1.7 points respectively. Brown clusters bring our parser (*main + brown*) less gain (0.2 points versus 0.7 points) than *tweebo - PTB*. Similar results are also observed when trained together with PTB data, and it is surprising to see that Brown clusters even harm the performance of Tweeboparser comparing *tweebo - brown* and *tweebo*. For tri-training experiments, as discussed before, it drastically improves our main model and the cascading model. With only 10k tri-training data, the main model has already outperformed Tweeboparser easily, and training on PTB data can give the cascading model another 0.4 points boost. In terms of speed, the main model is the fastest during decoding, over 15 times faster than Tweeboparser. The fast speed of our parser moves us closer

to making real world or even real time applications possible. We can also use more time complexed algorithms such as self-training to improve our parser. Meanwhile, it is not expected that Tweeboparser with and without PTB features gives us almost the same speed, because for each tweet input, Tweeboparser with PTB features parses it two times with different models while Tweeboparser without PTB features only needs one run. We think the reason for that is different models or feature sets of Tweeboparser could yield different convergence time of the dual decomposition algorithm. Clearly, although parsing with the PTB model takes extra time, the stacking of the PTB features accelerates the convergence of the tweet model.

On the whole, our main model and cascading model underperforms Tweeboparser only by a small margin, but with tri-training data, our parser can surpass Tweeboparser. We believe there are two main reasons for it. The first reason is that Tweeboparser is a graph-based parser using AD^3 algorithm (Martins et al., 2015) for global decoding, whereas our parser is a neural network version of greedy transition-based parser, where intermediate errors during the parsing process could largely influence the following decisions on the transition actions. Beam search could be a good option to integrate into our parser, but still we think the improvement will be minor and limited. The second reason, that we argue is the biggest obstacle for tweet parser, is the data limit. Tweebank training set is small and noisy, despite the fact that motivation of building Tweebank is a great idea. We can accelerate the annotation process hugely by introducing fewer guidelines and involving more people with less annotation experience, and the major complexity is handed over to the model itself, which we hope could learn to distinguish good and bad annotations and extract the correct knowledge in the training set. However, the knowledge Tweebank encodes is far from enough or salient for our neural tweet parser to learn well. Usually, neural network models need relatively larger scale of data to reach competitive results, and that either using PTB data or tri-training data can substantially improve our parser shows our neural model has the capability to absorb even larger amount of data. We believe that if our goal is to parse tweets, building a larger Twitter treebank with higher quality would be the priority

of our future direction. Another point that deserves discussion is that from experimental results, the evidence that Brown clusters are helpful seems not to be strong when training data becomes large. We observe that with small amount of training data, Brown clusters are much more helpful probably because they provide the hierarchical categoriazations of the words, functioning similar to POS tags. Such information is supplemented by increasing the size of training data, as the parameters of POS tags are better estimated. Moreover, when facing real world tweet data Brown clusters have bigger drawbacks, as OOV words will be encountered more frequently.

4.3 Summary

In this chapter, we experimented with different models and different configurations for our neural tweet parser. We investigate the effect of learnable word embeddings and Brown clusters on the input word representations. We found that Brown clusters are useful when training data scale is small but its importance becomes weaker when we access to more annotated data. We then tested whether the external data can be helpful for our neural tweet parser. We showed that the cascading model, after pre-trained on PTB data, performed considerably better than the main model. Although our main model and cascading model fall behind Tweeboparser by 3.8 and 0.8 UAS F-scores in the test set, both models are much faster than Tweeboparser, and the main model is over 15 times faster than Tweeboparser. By incorporating tri-training, we demonstrated that our neural tweet parser outperformed Tweeboparser by 0.8 and 1.2 points for the main model and cascading model respectively.

Chapter 5

CONCLUSIONS AND FUTURE WORK

In this thesis, we have attempted to tackle the problem of dependency parsing for tweets in two aspects, data and model.

We assume the syntactic structure of tweets comes from two parts, the general English phenomena like newswire and the Twitter specific linguistic phenomena. We investigated Tweebank (Kong et al., 2014) and analyzed the Twitter specific linguistic phenomena. We followed the same annotation conventions and tools as Kong et al. (2014), and created a new development set with 210 tweets for Tweebank. It has been proved to have consistent annotations with Tweebank test set, which is highly indicative when tuning hyperparameters on it.

We presented a novel neural dependency parser for tweets, neural tweet parser. We extended the stack LSTM parser to allow for Twitter specific linguistic phenomena. We experimented with different input word representations and show that the character-based architecture with pre-trained language model embeddings gives us the best balance between parsing performance and speed. We explored external data to transfer knowledge using PTB data and increase tweet data size using tri-training. Our cascading model pre-trained on PTB data performed better than the main model of neural tweet parser, and with augmented tri-training data, our parser outperformed Tweeboparser. In addition to this, our parser is over 15 times faster than Tweeboparser, which makes tweet parsing more practical as a component in future applications

We think the main problem for tweet parsing is the limit of the data in both quality and quantity. We showed that by simply incorporating either out-of-domain data or in-domain data, the performance of our parser can be improved. We found that the training data of

Tweebank lacks salient syntactic information on both general linguistic structures and Twitter specific structures. In the future, we would like to concentrate on the following methods in order to access to more data with information on both types of linguistic structures:

- Build a larger tweet treebank with high quality. This is the most direct way of both refining our syntactic analysis of tweets and improving the parser. We hope to engage more researchers in the community, who are interested in NLP for informal text with the project. Our goal is to create an updated version of Tweebank whose size is comparable to those of some popular low resource languages.

- Make better use of transfer learning on out-of-domain data like PTB.

- Explore massive unannotated tweet data for generating more training data. The fast speed of our parser makes algorithms like self-training possible compared to Tweeboparser.

BIBLIOGRAPHY

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D/D15/D15-1041`.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, December 1992. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=176313.176316`.

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*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

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 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 334–343, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1033`.

Jacob Eisenstein. What to do about bad language on the internet. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 359–369, Atlanta, Georgia, June 2013.

Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N13-1037.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. #hardtoparse: POS tagging and parsing the twitterverse. In *Analyzing Microtext, Papers from the 2011 AAAI Workshop, San Francisco, California, USA, August 8, 2011*, 2011a. URL http://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/view/3912.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. From news to comment: Resources and benchmarks for parsing the language of Web 2.0. In *Fifth International Joint Conference on Natural Language Processing, IJCNLP 2011, Chiang Mai, Thailand, November 8-13, 2011*, pages 893–901, 2011b. URL http://aclweb.org/anthology/I/I11/I11-1100.pdf.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *J. Mach. Learn. Res.*, 3:115–143, March 2003. ISSN 1532-4435. doi: 10.1162/153244303768966139. URL http://dx.doi.org/10.1162/153244303768966139.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 42–47, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-88-6. URL http://dl.acm.org/citation.cfm?id=2002736.2002747.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. A universal framework for inductive transfer parsing across multi-typed treebanks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 12–22, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL http://aclweb.org/anthology/C16-1002.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Mark Johnson. Pcfg models of linguistic tree representations. *Comput. Linguist.*, 24(4): 613–632, December 1998. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=972764.972768.

Jason Katz-Brown, Slav Petrov, Ryan McDonald, Franz Och, David Talbot, Hiroshi Ichikawa, Masakazu Seno, and Hideto Kazawa. Training a parser for machine translation reordering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 183–192, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-937284-11-4. URL http://dl.acm.org/citation.cfm?id=2145432.2145454.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A. Smith. A dependency parser for tweets. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1001–1012, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1108.

Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P08-1068.

Zhenghua Li, Min Zhang, and Wenliang Chen. Ambiguity-aware ensemble training for semi-supervised dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 457–467, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-1043.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=972470.972475.

A. Martins, M. A. T. Figueiredo, P. Aguiar, N. A. Smith, and E. P. Xing. Ad3: Alternating directions dual decomposition for map inference in graphical models. *Journal of Machine Learning Research*, 16(Mar):495–545, March 2015. ISSN 1532-4435.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 152–159, Stroudsburg, PA, USA, 2006a. Association for Computational Linguistics. doi: 10.3115/1220835.1220855. URL http://dx.doi.org/10.3115/1220835.1220855.

David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 337–344, Stroudsburg, PA, USA, 2006b. Association for Computational Linguistics. doi: 10.3115/1220175.1220218. URL http://dx.doi.org/10.3115/1220175.1220218.

T. Michael Mordowanec, Nathan Schneider, Chris Dyer, and A. Noah Smith.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios

Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

Truc-Vien T. Nguyen, Alessandro Moschitti, and Giuseppe Riccardi. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pages 1378–1387, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-63-3. URL `http://dl.acm.org/citation.cfm?id=1699648.1699684`.

Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, IncrementParsing '04, pages 50–57, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

Olutobi Owoputi, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *In Proceedings of NAACL*, 2013.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 705–713, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1870658.1870727`.

Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1524–1534, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-937284-11-4. URL http://dl.acm.org/citation.cfm?id=2145432.2145595.

Ivan A. Sag, Timothy Baldwin, Francis Bond, Ann A. Copestake, and Dan Flickinger. Multiword expressions: A pain in the neck for NLP. In *Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '02, pages 1–15, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43219-1. URL http://dl.acm.org/citation.cfm?id=647344.724004.

Nathan Schneider, Brendan O'Connor, Naomi Saphra, David Bamman, Manaal Faruqui, Noah A. Smith, Chris Dyer, and Jason Baldridge. A framework for (under)specifying dependency syntax without overloading annotators. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 51–60, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-2307.

Anders Søgaard and Christian Rishøj. Semi-supervised dependency parsing using generalized tri-training. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1065–1073, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1873781.1873901.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043064.

L Tesnière. *Elements de syntaxe structurale*. Editions Klincksieck, 1959.

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, Beijing, China, July 2015. Association for Computational Linguistics.

Jun Xie, Haitao Mi, and Qun Liu. A novel dependency-to-string model for statistical machine translation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 216–226, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D11-1020`.

H. Yamada and Y. Matsumoto. Statistical Dependency Analysis with Support Vector machines. In *The 8th International Workshop of Parsing Technologies (IWPT2003)*, 2003.

Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Trans. on Knowl. and Data Eng.*, 17(11):1529–1541, November 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.186. URL `http://dx.doi.org/10.1109/TKDE.2005.186`.