POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Master of Science in Computer Engineering

Master's Thesis

Evaluating Globally Normalized Transition Based Neural Networks for Multilingual Natural Language Understanding

Examiner:
prof. Paolo Garza

Candidate:
Andrea Azzarone

Supervisors:
prof. Olov Engwall
prof. Lluís Padró

April 2018

Abstract

We analyze globally normalized transition-based neural network models for dependency parsing on English, German, Spanish, and Catalan. We compare the results with FreeLing, an open source language analysis tool developed at the UPC natural language processing research group. Furthermore we study how the mini-batch size, the number of units in the hidden layers and the beam width affect the performances of the network. Finally we propose a multi-lingual parser with parameters sharing and experiment with German and English obtaining a significant accuracy improvement upon the monolingual parsers. These multi-lingual parsers can be used for low-resource languages or for all the applications with low memory requirements, where having one model per language is intractable.

Acknowledgement

I would like to show my greatest appreciation to Prof. Lluís Padró whose comments and suggestions were innumerably valuable throughout the course of my study. Special thanks also go to Prof. Olov Engwall and Prof. Paolo Garza for their support. Thanks to Canonical LTD that allowed me to work and study at the same time, to all my colleagues and my managers for their support and for all the teachings, a big hug here goes to Marco. I am deeply grateful to all the friends that shared this amazing journey with me. Thanks to Vicho, Berni (and Kala) for being the best roommates ever. Thanks to my brothers Giovanni e Domenico for being there when needed. Last but not least I would also like to express my gratitude to my parents for their support and warm encouragements.

Contents

List of Figures

List of Tables

Abbreviations

ACC Label Accuracy. 21

CRF Conditional Random Field. 13, 17

LAS Labeled Attachment Score. 21, 22

MTL Multi-Task Learning. 30

NLP Natural Language Processing. 1, 5, 9, 13, 31, 38

NLU Natural Language Understanding. 1, 5

PoS Part Of Speech. 3–5, 9–11, 19, 20, 22

UAS Unlabeled Attachment Score. 6, 21, 27, 34, 35

Chapter 1

Introduction

1.1 Background

Natural Language Processing (NLP) refers to the use and capacity of systems to process sentences in a natural language, for any purpose, independent of the level of in-depth analysis. Natural language means the language we use in everyday life, such as English, Spanish, Chinese, and Italian. It is synonymous with human language, mainly to distinguish it from formal language, including computer language. As it is, natural language is the most natural and common form of human communication. Compared to formal language, natural language is much more complex, it often contains misunderstandings and ambiguities, which makes it very difficult to elaborate. Interest in NLP began in 1950 with the publication of Alan Turing's paper "Computing Machinery and Intelligence" and ever since it exponentially increased: machine translations, predictive typing, spell checkers, and Spam detections are just few of everyday tools using NLP as foundation.

Natural Language Understanding (NLU) is a branch of NLP in artificial intelligence that studies machine reading comprehension. NLU is considered an hard problem. One of the main obstacle that makes NLU so difficult is that human languages exhibit notable degrees of ambiguity. Natural language grammars are ambiguous and common phrases have numerous potential analyses. Indeed for a normal sentence there may be thousands of probable parses (most of which will appear absolutely senseless to us).

Dependency-based parsers can be used to resolve such ambiguities. In recent years there has been an increasing interest in neural network models for dependency parsing. These models present all advantages and disadvantages that neural networks models usually present: training these types of parsers does not require a lot of expertise, but to obtain state-of-art results a considerable amount of annotated data is required.

On March 2016 Google published a globally normalized transition-based neural network model (SyntaxNet) obtaining state-of-the-art results in part-of-speech tagging and dependency parsing, among others (Andor *et al.*, 2016).

1.2 Intended readers

In this thesis, dependency parsing and neural networks models for dependency parsing are the main topics. A reader interested in either of these topics may find the results presented useful for further study. A casual reader will be presented with an introduction to dependency parsing, neural networks model for dependency parsing and SyntaxNet, the model that we aim to study and extend.

1.3 This thesis

We aim to understand how well globally normalized transition-based neural networks for dependency parsing perform on four different languages (English, German, Spanish and Catalan) and how the choice of parameters affects the performances.

We also aim to understand if you can somehow exploiting the similarities among two or more languages to train more robust models, jointly training a dependency parser for two languages at the same time. We also try to understand if the same methodology can be used for low resource languages, for which small annotated corpora are available.

We consider worthwhile this work as to our knowledge there is not yet an extensive study of SyntaxNet's parameters. This study can be used by the research community or by anyone else interested in training a SytaxNet model to tweak the parameters and to avoid a time-expensive parameters search to improve the accuracies. We also believe that the work on the bilingual parser can be used to get competitive accuracies for low-resource languages or in all the cases where we need to deal with more than one language and we cannot afford to waste memory and storage resources. It is indeed clear that having one model for language is more resources-demanding than having a single multi-lingual model. Resources used not to be a problem when inference was done on powerful systems, but nowadays we see more and more interest in performing inference directly in embedded devices.

The main contributions of this thesis are: an extended study on how SyntaxNet's parameters affect the performance of the model, and a novel multilingual parser based on SyntaxNet and trained on more languages.

1.3.1 Delimitations and Limitations

Considering the limited amount of computations resources available we limit the study of SyntaxNet to four languages: English, German, Spanish and Catalan. We also limit the study of the multilingual parser to two languages, German and English, performing a reduced but still valid training, as explained in more details in Chapter 8.

1.3.2 Related works

To our knowledge this is the first extended study of SyntaxNet's parameters. Even if Andor *et al.* (2016) extensively introduce SyntaxNet and its background and motivation, the presented results are limited to a small set of parameters.

On the other side this is not the first attempt to build a multilingual parser. Zeman and Resnik (2008) use delexicalized parsing. A delexicalized parser is a parser trained without any lexical features using the treebank of resource-rich source language. The delexicalized parser is then used to analyze phrases in the target resource-poor language. This take advantage of the idea that identical POS tags are highly descriptive of binary relations, and that many languages shares some semantic structure. Petrov *et al.* (2016) and Ma and Xia (2014) take advantage of parallel corpora as a way to project constraints from the source language to the target language. Duong *et al.* (2015) proposed a model for training a bilingual parser exploiting the idea of an interlingual continuous representation of language using a bilingual dictionary as a set of soft constraints.

Our approach is similar to the one of Duong *et al.* (2015) but we use pretrained multilingual word-embeddings instead of a bilingual dictionary, and we use as a baseline model SyntaxNet.

1.3.3 Ethical Concerns and Sustainability

Even if artificial intelligence is raising more and more ethical questions in the last few years, we cannot see any ethical concern in this thesis.

1.3.4 Outline of the thesis

From Chapter 2 till Chapter 5 we progressively introduce the reader to Dependency Parsing, Transition-Based Dependency Parsing, Neural Network Models for Dependency Parsing and SyntaxNet.

The last two chapters contains the main contributions of our thesis. In Chapter 6 we describe how we trained several SyntaxNet models for multi-lingual dependency parsing. In Chapter 7 we study how the choice of SyntaxNet's parameters affects the performances of the models. In Chapter 8 we describe our experiments to build and train jointly bilingual parsers based on the idea that we can exploit similarities between languages.

Chapter 2

Dependency Parsing

Dependency-based syntactic theories are built around the theory that the syntax of a sentence can be expressed as a set of relations between the words of the sentence (Nivre, 2008). Dependency parsing demonstrated its value in several applications such as Machine Translation (Chris Quirk, 2006; Ding and Palmer, 2005; Shen *et al.*, 2008), Question Answering (Jijkoun *et al.*, 2004; Cui *et al.*, 2005a; Bouma *et al.*, 2006; Cui *et al.*, 2005b), Information Extraction (Ciravegna and Lavelli, 1999; Sætre *et al.*, 2008; Alphonse *et al.*, 2004) and Natural Language Generation (Koller and Striegnitz, 2002) among others.

2.1 Dependency Graph

The syntactic structure of a sentence is expressed by a dependency graph as illustrated in Figure 2.1 for an English sentence taken from the Penn Tree-bank (Marcus *et al.*, 1993). It's common practice to add a bogus token `ROOT` at the beginning of the sentence, acting as the unique root of the graph. More formally:

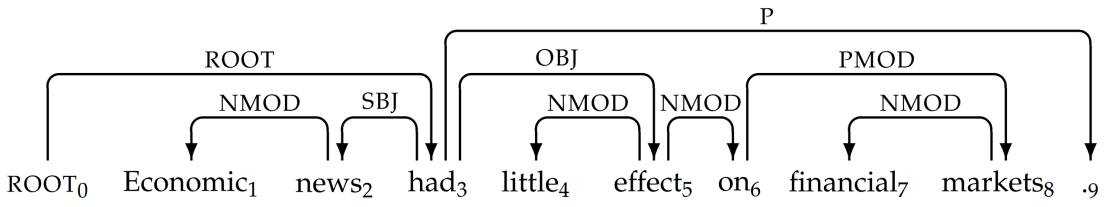


Figure 2.1: *Dependency graph for an English sentence from the Penn Tree-bank (source (Gómez-Rodríguez and Nivre, 2013)).*

Definition 2.1.1. Identified a set $\boldsymbol{L} = \{l_1, \ldots, l_{|L|}\}$ of dependency labels, a **dependency graph** for a sentence $x = \{w_0, w_1, \ldots, w_n\}$ is a labeled directed graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A})$, where

1. $\boldsymbol{V} = \{0, 1, \ldots, n\}$ is a set of nodes and each node is a non-negative integer corresponding to the index of a word in the sentence (including `ROOT`).
2. $\boldsymbol{A} \subseteq \boldsymbol{V} \times \boldsymbol{L} \times \boldsymbol{V}$ is a set of labeled directed arcs. Each labeled arc can be expressed as an ordered triple (i, l, j) where i and j are nodes and l is a dependency label. The node i can be referred to as the **head**, l as the **dependency type** of j, and j as a **dependent** of i (Nivre, 2008).

Definition 2.1.2. A dependency graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A})$ is **well-formed** on these terms:

1. The node 0 is a root.
2. Every node has just no more than one head and one label.
3. The graph $\boldsymbol{G}$ has no directed cycles.

A well-formed dependency graph (satisfying all the above conditions) is a **dependency forest**. A connected dependency forest is also called **dependency tree**.

2.2 Dependency Parsers

In computational linguistics, a **dependency parser** is a device that produces a dependency graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A})$ from an input sentence $x = w_0, \ldots, w_n$ and eventually other inputs, e.g. PoS tags as illustrated in Figure 2.2.

Figure 2.2: *Dependency parser schema*

In the last few years dependency-based syntactic representations have been used mostly in data-driven models, which learn to parse sentences for syntactic relations merely from an annotated corpus. One potential advantage of these type of models is that they are handily ported to any language in which annotated corpora subsist.

In the course of our work we made use of two dependency parsers, SyntaxNet and FreeLing. SyntaxNet is an open-source neural network framework implemented in TensorFlow that provides a base for NLU systems (Andor *et al.*, 2016). We'll talk more about it in Chapter 5.

FreeLing is an open source language analysis tool suite created and currently leaded by Lluís Padró as a tool to make easily available the results of the research conducted at the Polytechnic University of Catalonia NLP research group (Padró and Stanilovsky, 2012). FreeLing implements both a command-line interface and a C++ library that can be used to parse text and obtain output in the several format (XML, JSON, and CoNLL) for several languages (English, Spanish, Italian, Catalan, and German among others). For dependency parsing FreeLing offers both rule-based and statistical models. The statistical one based on (Carreras, 2007) will be used in this thesis from now on. The statistical dependency parser uses the Treeler[1] machine learning library.

[1] `http://devel.cpl.upc.edu/treeler/`

Chapter 3

Transition-Based Dependency Parsing

Transition-based dependency parsers scan a sentence word by word, and sequentially execute transition actions to build the parse tree of the sentence(Nivre, 2008). Transition-based dependency parsers are defined in terms of two components, a **transition system** and an **oracle**.

3.1 The Transition System

Definition 3.1.1. A **transition system** for dependency parsing consist of:

1. a set of configuration (or parser states) $\boldsymbol{C}$,
2. a set of rules (or transitions) $\boldsymbol{T}$ for which configuration to go from each configuration,
3. a set of initial configurations
4. the set of terminal configurations $\boldsymbol{C_t} \subseteq \boldsymbol{C}$

A configuration has to contain a buffer $\boldsymbol{\beta}$, initially containing the nodes $[1, \ldots, n]$ matching the words of a phrase $x = (w_0, w_1, \ldots, w_n)$, and a set $\boldsymbol{A}$ of dependency arcs. Real-word transition systems enhance this essential definition of configuration with various data structures, such as list and stacks. We use the notation $\boldsymbol{\beta_c}$ and $\boldsymbol{A_c}$ to refer to the value of $\boldsymbol{\beta}$ and $\boldsymbol{A}$, respectively, in a configuration c. An empty buffer is denoted using [].

Definition 3.1.2. Given a transition system $\boldsymbol{S}$, A **transition sequence** for a sentence $x = (w_1, \ldots, w_n)$ in $\boldsymbol{S}$ is a sequence $\boldsymbol{C_{0:m}} = (c_0, c_1, \ldots, c_m)$ of states, so that:

1. c_0 is an initial state
2. c_m is a terminal state, $c_m \in \boldsymbol{C_t}$
3. for every i ($1 \leq i \leq m$), $c_i = t(c_{i-1})$ for some $t \in \boldsymbol{T}$.

The **parse** assigned to the sentence x by the transition sequence $\boldsymbol{C_{1:m}}$ is the dependency graph $\boldsymbol{G_{c_m}} = (\{1, \ldots, n\}, \boldsymbol{A_{c_m}})$. Starting from the initial parser state, transitions will manipulate the buffer $\boldsymbol{\beta}$ and the set of dependency arcs $\boldsymbol{A}$ (and other available data structures) until a terminal configuration is hit. The set $\boldsymbol{A_{c_m}}$ of dependency arcs in the terminal parser state will determine the output dependency graph $\boldsymbol{G_{c_m}} = (\boldsymbol{V}, \boldsymbol{A_{c_m}})$. We can consider a

Table 3.1: *An example of transition-based dependency parsing using the arc-standard rules.*

Transition	Stack	Buffer	A
	[ROOT]	[She has bad taste .]	
SHIFT	[ROOT She]	[has bad taste .]	
SHIFT	[ROOT She has]	[bad taste .]	
LEFT-ARC (nsubj)	[ROOT has]	[bad taste .]	A[nsubj(has, She)]
SHIFT	[ROOT has bad]	[taste .]	
SHIFT	[ROOT has bad taste]	[.]	
LEFT-ARC (amod)	[ROOT has taste]	[.]	A[amod(taste, bad)]
RIGHT-ARC (dobj)	[ROOT has]	[.]	A[dobj(has, taste)]
SHIFT	[ROOT has .]	[]	
RIGHT-ARC (punct)	[ROOT has]	[]	A[punct(has, .)]
RIGHT-ARC (root)	[ROOT]	[]	A[root(ROOT, has)]

bijective correspondence between decision sequences $\boldsymbol{t_{0:j-1}}$ and configurations c_j: in other words, we consider that the whole history of decisions is encoded by a configuration and each configuration can reached by a unique decision sequence from c^0.

Two of the most common used transition system for dependency parsing are the arc-standard and the arc-eager.

3.1.1 Arc-standard Transition System

In the arc-standard a state $c = (\boldsymbol{s}, \boldsymbol{b}, \boldsymbol{A})$ subsists of a *buffer* $\boldsymbol{b}$, a *stack* $\boldsymbol{s}$, and a set of *dependency arcs* $\boldsymbol{A}$. For an input sentence $w_1, \ldots, w_n$ parsing starts with $\boldsymbol{s} = [\texttt{ROOT}]$, $\boldsymbol{b} = [w_1, \ldots, w_n]]$, and $\boldsymbol{A} = \emptyset$. On the contrary parsing stops if the buffer is empty and the stack only contains the node `ROOT`. At each step one of the following *transition action* can be taken:

- SHIFT: pushes the front token b_1 from the buffer onto the stack.
- LEFT-ARC(l): insert an arc $s_1 \rightarrow s_2$ with label l and pops s_2 from the stack.
- RIGHT-ARC(l): insert an arc $s_2 \rightarrow s_1$ with label l and pops s_1 from the stack.

Table 3.1 illustrates an example of a transition sequence from the initial configuration to a terminal one for the sentence *"She has bad taste"*.

3.1.2 Arc-Eager Transition System

In the arc-eager, a state $c = (\boldsymbol{s}, \boldsymbol{b}, \boldsymbol{A})$ subsist of a *buffer* $\boldsymbol{b}$, a *stack* $\boldsymbol{s}$, and a set of *dependency arcs* $\boldsymbol{A}$. For an input sentence $w_1, \ldots, w_n$ parsing starts with $\boldsymbol{s} = [\texttt{ROOT}]$, $\boldsymbol{b} = [w_1, \ldots, w_n]$, and $\boldsymbol{A} = \emptyset$. On the contrary parsing stops if the buffer is empty. At each step one of the following *transition action* can be taken:

- SHIFT: pushes the front token b_1 from the buffer onto the stack.
- LEFT-ARC(l): insert an arc $w_0 \rightarrow s_0$ with label l and pops s_0 from the stack.
- RIGHT-ARC(l): insert an arc $s_0 \rightarrow w_0$ with label l and pops w_0 onto the stack.

- REDUCE: pops the stack.

In the arc-eager system arcs are added at the earliest possible chance. It consequently produces parts of the tree top-down rather than bottom-up. It does not guarantee well-formedness.

3.2 The Oracle

Definition 3.2.1. An **oracle** for a transition system $\boldsymbol{S} = (\boldsymbol{C}, \boldsymbol{T}, c_s, \boldsymbol{C_t})$ is a function $o : \boldsymbol{C} \rightarrow \boldsymbol{T}$.

Deterministic parsing can be implemented as in Algorithm 1. It's straightforward to see that a deterministic parser builds one and only one transition sequence $\boldsymbol{C_{0:m}}$ for a sentence x. The terminal configuration c_m defines the final parse tree.

Algorithm 1: Oracle parsing algorithm

Data: $x = (w_0, w_1, \ldots, w_{n-1})$
Result: G_c
$c \leftarrow c_s$;
while $c \notin C_t$ **do**
  $c \leftarrow [o(c)](c)$
end

In data-driven dependency parsing, oracles normally take the form of classifiers (Nivre, 2008). In the last few years oracles have been usually implemented using neural networks.

Over-generalizing we can classify them as:

- **local** - e.g. at each iteration a set of successor states are produced, a classifier (e.g. a SVM) is then used to predictably pick the one with the highest score as the next state so behaving as a local classifier.
- **structured** - the oracle optimizes whole sequences of actions. They're called structured because full sequences of actions corresponds to tree structures

Chapter 4

Neural Network Models for Transition-Based Dependency Parsing

Until few years ago, transition-based dependency parsers described in Chapter 3 were implemented using rule-based oracles. The use of many feature templates, required in these types of parsers, creates several problems: first of all because the feature templates have to be manually designed, a lot of know-how is required; even with a huge amount of training data is required, it's almost impossible to train all feature weights; last but not least, the time required to extract the feature is usually higher than the time required by the main parsing algorithm Chen and Manning (2014),

Neural network models for transition-based dependency parsing address all the these problems. Chen and Manning (2014) use dense features in place of the sparse indicator features as inspired by the success of distributed word representations in several NLP tasks. Furthermore a neural network classifier is trained to make parsing decisions. The neural network learns dense vector representation of words, dependency labels, and part of speech tags. Chen and Manning (2014) introduced for the first time PoS tag and arc label embeddings in place of discrete representations.

In the next few pages we are going to give a short introduction to word embeddings, what they are and why in the few past years there has been so much interest in them. Furthermore we'll briefly describe the greedy and the structured architectures.

4.1 Word Embeddings

Until few year ago, the standard way to encode sentences and works was the Bag-of-Words. With this model each word is encoded as one-hot vector. The dimensionality of the vector is equal to the size of the vocabulary. So let's consider a 6-words vocabulary: *king*, *queen*, *man*, *woman*, *uncle*, and *aunt*. The representation for each word in the vocabulary is respectively $\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}$, and $\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}$.

This model presents several issues:

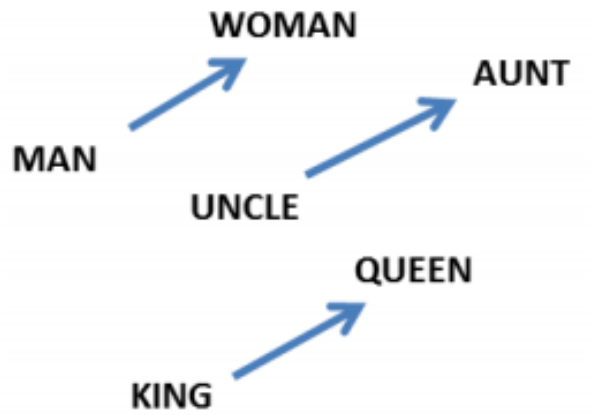


Figure 4.1: *Example of encoded semantic relationships in word embeddings.*

- each word vector has no semantical meaning. That's to say there is no relationship between vectors that encode similar words (e.g. king and queen, man and woman)
- in real-word scenarios, the vocabulary size can be huge. This mean that the dimensionality of each word vector is huge and the possibilities to stumble in the curse of dimensionality are high.

Word embeddings (also referred to as *distributed representation*, *semantic vector space*, or *word space*) solve both these issues. The idea is to use a mathematical embedding to go from a one-dimension-per-word vector space to a continuous one with much lower dimension. Credit for the definitive spread of word embeddings goes to Mikolov *et al.* (2013) with the introduction of `word2vec`. It was shown that the similarity of word representations goes beyond simple syntactic regularities. For example that $vector(\text{"}Uncle\text{"}) - vector(\text{"}Man\text{"}) + vector(\text{"}Woman\text{"})$ results in a vector that is close to the vector embedding of the word $Aunt$ as shown in picture 4.1.

4.1.1 PoS and Label Embeddings

Although the PoS tags and arc labels are relatively small discrete sets, they still manifest many semantical similarities like words. For instance, `NN` (singular noun) must be closer to `NNP` (Proper Noun) than `JJ` (Adjective), and `amod` (adjective modifier) must be closer to `poss` (possession modifier) than `dobj` (direct object). As first shown by Chen and Manning (2014), the learned dense representations of PoS tags and arc labels correctly carry these semantical similarities. Figure 4.2 presents t-SNE visualizations of these embeddings.

4.2 Greedy Neural Network Model

Figure 4.3 describes the neural network architecture introduced by Chen and Manning (2014) that still serves as baseline for more recent works, including SytanxNet. As stated above each word, PoS tag and arc label is represented as a d-dimensional vector. A set of elements are chosen in accordance with the stack/buffer position for each type of information (word, PoS or arc label). A benefit of this kind of parser is that we can easily add a rich set of elements, rather than manually defining many more indicator features.

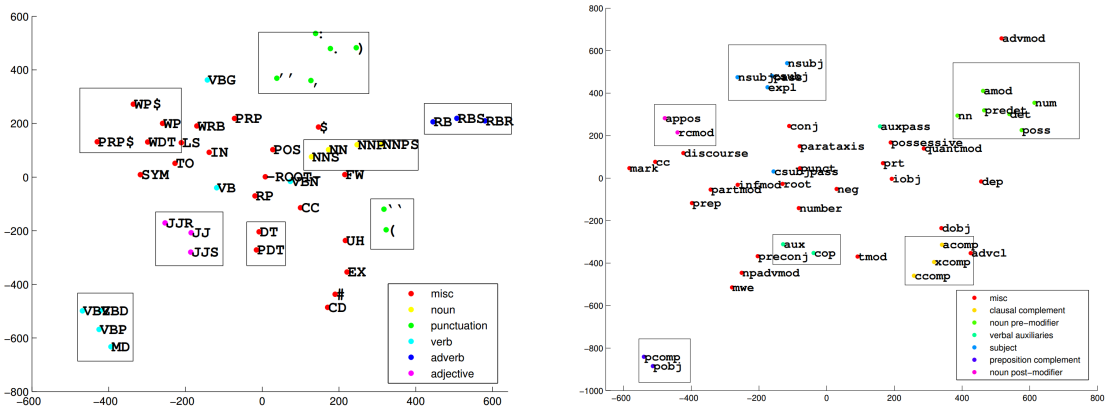


Figure 4.2: *t-SNE visualization of PoS and label embeddings. Similar PoS and labels are close in the embedding space. Source Chen and Manning (2014).*

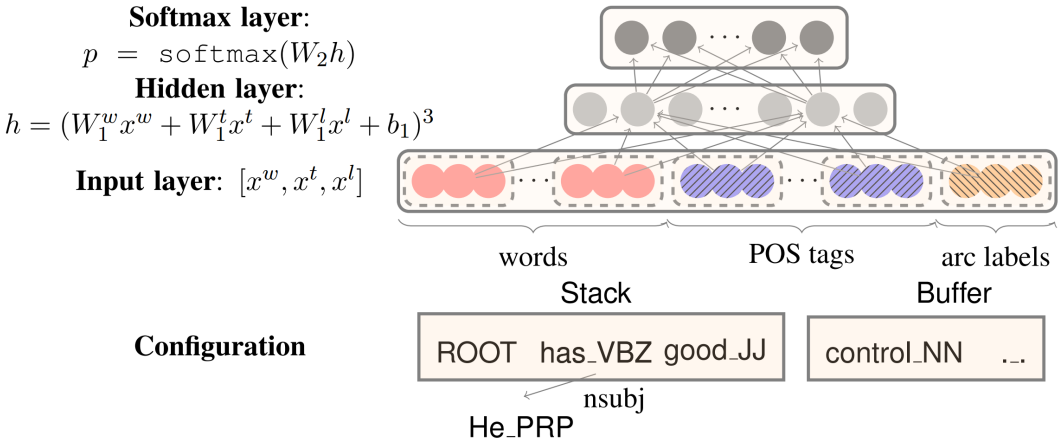


Figure 4.3: *Neural network architecture used in Chen and Manning (2014) .*

A neural network with one hidden layer is built, where the corresponding embeddings of the chosen elements (words, PoS tags and labels) are added to the input layer. A cube activation function is used. Finally a softmax layer is used to model multi-class probabilities.

Training instances are produced from the training data and their parse trees adopting a deterministic oracle which always choose `LEFT-ARC` over `SHIFT`. The training objective is to minimize the cross-entropy loss. A l_2-regularization term is used to reduce overfitting.

At parsing time the model performs greedy decoding. At each step, all the corresponding word, PoS tag and label embeddings from the current configuration are extracted, the hidden layer activation is computed and the transition with highest score is picked.

4.3 Structured Neural Network Model

The downside of greedy (or local) parsing described above is error propagation: an incorrect action will have a negative impact on the following ones, resulting an incorrect output parse tree. To cope with this issue global structural learning can be used.

- In the aspect of decoding, beam-search can be applied to improve upon greedy one-best search.
- In the aspect of training, global structural learning can be used to replace local learning on each decision.

Beam search is a heuristic search algorithm that reduces the number of nodes to be visited pruning at any step all non-promising nodes (Zhang, 1999). The pruning is driven by problem-specific heuristics. The group of most promising nodes is named "beam" (Xu and Fern, 2007). The size of the beam is considered a parameter of the search, and can vary during the search process.

Global structural learning maximizes the likelihood of action sequences rather than the likelihood of individual actions. Not doing so means that at any search step we consider each action conditionally independent from the past and the future ones. Regarding dependency parsing, this is in general not true, hence a structured learning should improve the effectiveness of a model.

Structured learning means predicting structured objects, instead of "simple" values. Here "simple" implies regression, binary and multi-class classification. For instance, the problem of obtaining the parse tree of a phrase can be seen as a structured learning problem in which the output domain is the set of all potential parse trees.

One of the most common example of structured learning algorithms is the structured perceptron (Collins, 2002). The desired outcome is a mapping from inputs $x \in \boldsymbol{X}$ to outputs $y \in \boldsymbol{Y}$. Given:

- Training epochs $\boldsymbol{T}$
- Training data $(x_i, y_i)\ \forall i = 1 \ldots n$.
- A function GEN which generates a set of successors given an input x.
- A representation $\boldsymbol{\Phi}$ mapping each $(x, y) \in \boldsymbol{X} \times \boldsymbol{Y}$ to a feature vector $\boldsymbol{\Phi}(x, y) \in \boldsymbol{R^d}$.
- A parameter vector $\vec{w} \in \boldsymbol{R^d}$.

The mapping from an input x to an output $F(x)$ is defined in such a way:

$$F(x) = \underset{y \in GEN(x)}{\arg\max}\ \boldsymbol{\Phi}(x, y) \cdot \vec{w}$$

Algorithm 2 and training data can be used to learn the parameter vector $\vec{w}$.

Algorithm 2: The structured perceptron learning algorithm

Data: training examples (x_i, y_i)
Result: $\vec{w}$

$\vec{w} \leftarrow 0$;
for $t \leftarrow 1$ ***to*** T **do**
  for $i \leftarrow 1$ ***to*** N **do**
    $z_i \leftarrow \arg\max_{z \in GEN(x)} \Phi(x_i, z) \cdot \vec{w}$;
    if $z_i \neq y_i$ **then**
      $\vec{w} \leftarrow \vec{w} + \Phi(x_i, y_i) - \Phi(w_i, z_i)$;
    end
  end
end

The structured perceptron has been successfully applied to several NLP tasks. For example for dependency parsing the most common used algorithm used is shown in Algorithm 3. On each training example, beam search ($DECODE$) is used to decode the sentence until the gold-standard parse tree falls out of the beam. As stated before beam search requires a score function for each parser state. This is usually calculated running a feed-forward network with the state as input (that's to say the set of features that describe each parser state). The last layer (the soft-max) of the feed-forward network is usually the distinguishing feature: e.g. Zhou *et al.* (2015) use a contrastive probabilistic learning objective, and Andor *et al.* (2016) introduce a Conditional Random Field (CRF) objective. The $UPDATE$ function is normally a stochastic gradient step taken on the particular objective.

Weiss *et al.* (2015) do not use a stochastic gradient step for the "update" function neither a special soft-max for decoding, instead a feed-forward network is first trained as by Chen and Manning (2014) and then a perceptron layer is trained using the standard structured perceptron algorithm where the representation function $\mathbf{\Phi}$ is computed by the trained feed-forward network as show in Figure 4.4.

Algorithm 3: Training Algorithm for Structured Neural Parsing

Data: training examples $(\boldsymbol{X}, \boldsymbol{Y})$
Result: θ
initialize θ;
for $t \leftarrow 1$ ***to*** T **do**
  foreach $i \leftarrow 1$ ***to*** N **do**
    $beam = \emptyset$;
    $goldState = null$;
    $terminate = null$;
    $beamGold = true$;
    while $beamGold$ ***and not*** $terminate$ **do**
      $beam = DECODE(beam, x_i, z_i)$;
      $goldState = GOLD\_MOVE(goldState, x_i, z_i)$;
      if ***not*** $IS\_GOLD(beam)$ **then**
        $beamGold = false$;
      end
      if $ITEMS\_COMPLETE(beam)$ **then**
        $terminate = true$;
      end
    end
    $\theta = \theta + UPDATE(goldState, beam)$;
  end
end

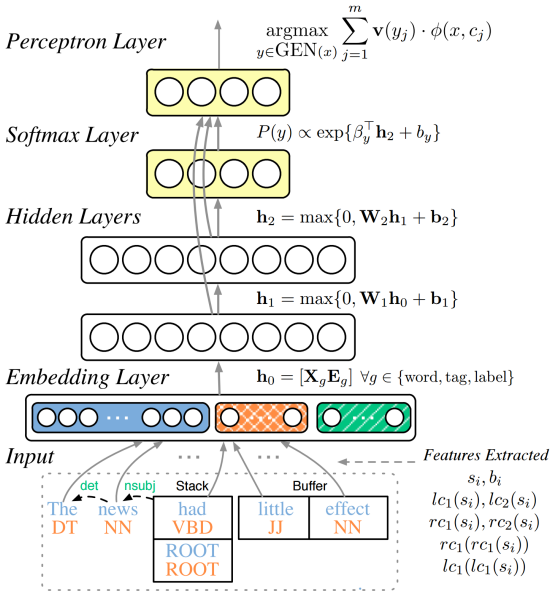


Figure 4.4: *Schematic overview of Weiss et al. (2015) neural network model.*

Chapter 5

SyntaxNet

In Chapter 4 we introduced the concept of structured neural network models for transition based parsing along with the abstract training algorithm. The beam search heuristic requires to score each parser state. The scoring function can be defined and implemented in several ways. Following Zhou *et al.* (2015) and Weiss *et al.* (2015), Andor *et al.* (2016) defines it via a feed-forward neural network as:

$$p(s, d; \theta) = \mathbf{\Phi}(s; \theta^{(l)}) \cdot \theta^{(d)}$$

where:

- $\theta^{(l)}$ are the parameters of the neural network, not including the parameters of the terminal layer.
- $\mathbf{\Phi}(s; \theta^{(l)})$ is the representation for state s computed by the model using the parameters $\theta^{(l)}$.
- $\theta^{(d)}$ are the terminal layer parameters for the decision d.

We can observe that the above-defined score is linear in the parameters $\theta^{(d)}$. The different models in the literature usually share the first layers of the feed-forward neural network but differ in the choice of the last layer, normally a softmax-style one. In the next few subsections we'll briefly describe the architecture of SyntaxNet shown in Figure 5.1.

5.1 The model

5.1.1 Input Layer

From a parse configuration c (consisting of a stack s and a buffer b), a set of discrete features F (grouped by category e.g. words, arc labels, and part of speech tags), is obtained and fed into the network. Once the features are obtained a $F \times V$ matrix X is constructed: the element X_{fv} is 1 if the f-th feature has value v, 0 otherwise. Please notice that F is the number of features and V is the size of the vocabulary of the feature group. For all groups, additional "tokens" are used:

- ROOT pointing out the part-of-speech or word of the root token.
- UNK pointing out an out-of-vocabulary token.
- NULL pointing out that no valid feature value could be determined.

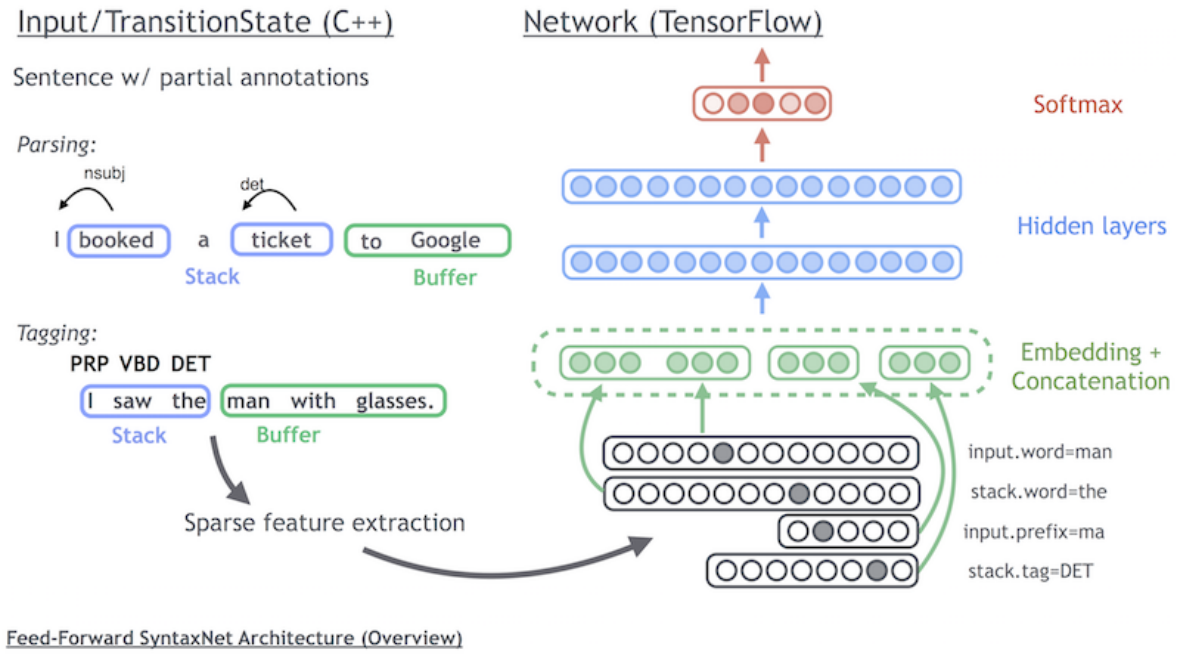


Figure 5.1: *Feed-Forward SyntaxNet architecture (source Google).*

5.1.2 Embedding Layer

The first learned layer h_0 converts the sparse features $\boldsymbol{X}$ into an embedded representation. For each feature group $\boldsymbol{X_g}$, we learn a $V_g \times D_g$ embedding matrix $\boldsymbol{E_g}$ that applies the conversion:

$$\boldsymbol{h_0} = [\boldsymbol{X_g}\boldsymbol{E_g}|g \in word, tag, ...]$$

the computation is applied separately for each group g and the results concatenated. The embedding dimensionality D can be chosen freely for each group.

5.1.3 Hidden Layers

Two fully-connected layers of rectified linear units (ReLu) are used as hidden layers. This differs from Chen and Manning (2014) that used a novel cube activation function.

5.1.4 Global vs. Local Normalization

For greedy neural network parsing, the conditional probability distribution over decision d_j given context $d_{1:j-1}$ is defined as

$$p(d_j|d_{1:j-1};\theta) = \frac{\exp p(d_{1:j-1}, d_j;\theta)}{Z_L(d_{1:j-1};\theta)} \tag{5.1}$$

where

$$Z_L(d_{1:j-1};\theta) = \sum_{d' \in A(d_{1:j-1})} \exp p(d_{1:j-1}, d';\theta)$$

where $A(d_{1:j-1})$ is the set of allowed decisions from the state s_i. Each $Z_L(d_{1:j-1}$ is a *local* normalization term. The probability of a decisions-sequence $d_{1:n}$ is:

$$
\begin{aligned}
p_L(d_{1:n}) &= \prod_{j=1}^{n} p(d_j | d_{1:j-1}; \theta) \\
&= \frac{\exp \sum_{j=1}^{n} p(d_{1:j-1}, d_j; \theta)}{\prod_{j=1}^{n} Z_L(d_{1:j-1}; \theta)}
\end{aligned}
\tag{5.2}
$$

Beam search can be applied to maximize Equation 5.2 with respect to $d_{1:n}$.

On the contrary, a CRF defines a distribution $p_G(d_{1:n})$ in the following way:

$$
p_G(d_{1:n}) = \frac{\exp \sum_{j=1}^{n} p(d_{1:j-1}, d_j; \theta)}{Z_G(\theta)} \tag{5.3}
$$

where

$$
Z_G(\theta) = \sum_{d'_{1:n} \in D_n} \exp \sum_{j=1}^{n} p(d'_{1:j-1}, d'_j; \theta)
$$

and D_n is the set of all valid sequences of decisions of length n. $Z_G(\theta)$ is a *global* normalization term. The inference problem is to find:

$$
\arg\max_{d_{1:n} \in D_n} p_G(d_{1:n}) = \arg\max_{d_{1:n} \in D_n} \sum_{j=1}^{n} p(d_{1:j-1}, d_j; \theta)
$$

Beam search is used to approximately find the argmax.

5.2 Training

Training data consists of input x paired with gold decision sequences $d^*_{1:n}$. Averaged Stochastic gradient descent with momentum is used on the negative log-likelihood of the data under the model to update the parameters. Under a locally normalized model, the negative log-likelihood is

$$
\begin{aligned}
L_{local}(d^*_{1:n}; \theta) &= -\ln p_L(d^*_{1:n}; \theta) \\
&= -\sum_{j=1}^{n} p(d^*_{1:j-1}, d^*_j; \theta) + \sum_{j=1}^{n} \ln Z_L(d^*_{1:j-1}; \theta)
\end{aligned}
\tag{5.4}
$$

while under a globally normalized model it is:

$$
\begin{aligned}
L_{global}(d^*_{1:n}; \theta) &= -\ln p_G(d^*_{1:n}; \theta) \\
&= -\sum_{j=1}^{n} p(d^*_{1:j-1}, d^*_j; \theta) + \ln Z_G(\theta)
\end{aligned}
\tag{5.5}
$$

considering that the Z_G is in many case intractable, beam search and early updates are used. As per algorithm 3, in case the gold path falls off of the beam at step j, a stochastic gradient step is executed with the following objective:

$$
L_{global-beam}(d^*_{1:j}; \theta) = -\sum_{i=1}^{j} p(d^*_{1:i-1}, d^*_i; \theta) + \ln \sum_{d'_{1:j} \in B_j} \exp \sum_{i=1}^{j} p(d'_{1:i-1}, d'_i; \theta)
$$

B_j includes all paths in the beam at iteration j. In case the gold path persists in the beam once the decoding is over, a gradient step is executed using B_n, the beam at the end of decoding.

While directly optimizing the global model defined by Equation 5.5 works well, it has been seen that training the model in two steps gets the same precision much faster: in the first step the local objective from Equation 5.4 is adopted to pretrain the network; in the second step the the global objective from Equation 5.5 is used.

Chapter 6

Evaluation of SyntaxNet on Multilingual Tasks

We evaluated SyntaxNet on 4 different languages (Catalan, German, English and Spanish) and compared our models with FreeLing. Even if SyntaxNet can be used both as a PoS tagger and as a dependency parser, in our work we focused on the latter part. Please note that here we trained and evaluated SyntaxNet for each language separately. In chapter 8 we describe our experiments to train the model jointly for two languages.

6.1 Corpora

To train and evaluate the models we used a modified version of CoNLL datasets provided by the Polytechnic University of Catalonia. The corpora provided by UPC included just training and tuning datasets. In order to properly evaluate the trained models we randomly extracted from the training sets a set of sentences to be used during testing. The exact statistics are shown in Table 8.1.

Table 6.1: *Number of documents for dataset.*

	# training	# tuning	# test
English	37945	1334	1334
German	31591	1828	1828
Spanish	12651	1654	1654
Catalan	11414	1709	1709

6.1.1 EAGLES PoS Tags

The corpora provided by UPC does not use the Universal PoS tags. Instead it uses PoS tags based on the proposals by EAGLES.

EAGLES PoS tags consist of labels of variable length where the front character always indicates the category (PoS). The length of the tag and the meaning of each character in the tag is regulated by the category.

For example, we could have the following definition for the category *noun*:

Position	Attribute	Values
0	category	**N**:noun
1	type	**C**:common; **P**:proper
2	genre	**F**:feminine; **M**:masculine; **C**:common
2	number	**S**:singular; **P**:plural; **N**:invariable

This would allow PoS tags such as *NCFP* (standing for *noun/common/feminine/plural*). Features that are not applicable or undetermined are set to 0 (zero). E.g. *NCF0* stands for *noun/common/feminine/underspecified-number*. The meaning of a character at a certain position of a tag depends on the category (determined by the first letter) and on the target language. E.g. for languages where nouns can have supplementary features (case, etc.) the tag characterization would include one further character.

Even if SyntaxNet is flexible enough to accept different tag sets, the tag set choice can sensibility affect the performance of the system. For more information we refer the reader to FreeLing documentation[1].

6.2 Model Configuration and Training

6.2.1 Features

The adopted model configuration is essentially the same as the one proposed by Andor *et al.* (2016) as shown in Figure 5.1. The arc-standard transition system was used and two different sets of features were experimented with. In the first configuration we extracted words (X^w), part of speech tags (X^t), and dependency arcs and labels (X^l) in the surround context of the state as shown in Table 6.2. In the second configuration we also added morphological features (X^m) if available. We used the following embedding sizes: 64 for words and 32 for all the other features.

SyntaxNet uses a flexible markup language to define feature functions:

- **word** (w) - to get the word from the focus token.
- **tag** (t) - to get the POS tag from the focus token.
- **label** (l) - to compute the label from focus token.
- **morphology-set** (m) - to get the list of morphological features from the focus token.

And feature locators:

- **input** (i) - Accesses the remaining input tokens in the parser state.
- **stack** (st) - Accesses the stack in the parser state.

[1] `https://talp-upc.gitbooks.io/freeling-user-manual/content/tagsets.html`

Table 6.2: *Feature templates in the first and second configuration. The highlighted row is used just in the second configuration.*

	Templates
X^w	i.w i(1).w i(2).w i(3).w st.w st(1).w st(2).w st(3).w st.rc(1).w st.rc(1).ls(1).w st.lc(1).w st.lc(1).rs(1).w st(1).rc(1).w st(1).rc(1).ls(1).w st(1).lc(1).w st(1).lc(1).rs(1).w st.rc(2).w st.lc(2).w st(1).rc(2).w st(1).lc(2).w
X^t	i.t i(1).t i(2).t i(3).t st.t st(1).t st(2).t st(3).t st.rc(1).t st.rc(1).ls(1).t st.lc(1).t st.lc(1).rs(1).t st(1).rc(1).t st(1).rc(1).ls(1).t st(1).lc(1).t st(1).lc(1).rs(1).t st.rc(2).t st.lc(2).t st(1).rc(2).t st(1).lc(2).t
X^l	st.rc(1).l st.rc(1).ls(1).l st.lc(1).l st.lc(1).rs(1).l st(1).rc(1).l st(1).rc(1).ls(1).l st(1).lc(1).l st(1).lc(1).rs(1).l st.rc(2).l st.lc(2).l st(1).rc(2).l st(1).lc(2).l
X^m	i.m i(1).m i(2).m i(3).m st.m st(1).m st(2).m st(3).m st.rc(1).m st.rc(1).ls(1).m st.lc(1).m st.lc(1).rs(1).m st(1).rc(1).m st(1).rc(1).ls(1).m st(1).lc(1).m st(1).lc(1).rs(1).m st.rc(2).m st.lc(2).m st(1).rc(2).m st(1).lc(2).m

- **left child** (lc) - Locates left children of the focus token.
- **right child** (rc) - Same as above but it locates right children.
- **left sibling** (ls) - Locates left siblings of the focus token.
- **right sibling** (rs) - Same as above but it locates right siblings.

6.2.2 Training

The model is trained in two steps in order to speed up the training process (Andor *et al.*, 2016): in in the first step the local objective from Equation 5.4 is adopted to pretrain the network; in the second step the the global objective from Equation 5.5 is used. All layers (including the embeddings and except the soft-max) were pretrained in this way. Averaged stochastic gradient descent with momentum is used.

If not specified we used an initial learning rate of 0.07/0.02 (greedy/structured), a momentum of 0.85/0.9 and a decay rate that corresponds to about one tenth of the training corpus size. All the models were trained with early stopping for a maximum of 12/20 epochs. We did not feed pre-trained word-embeddings, instead we normally initialized and learnt them.

6.3 Results

Table 6.3 shows our final parsing test-set results and a comparison with FreeLing. The FreeLing models were trained with the exact same corpora used for our models. We provide the UAS, the Labeled Attachment Score (LAS) and the Label Accuracy (ACC) calculated using the CoNLL-2007 Shared Task Scorer:

- LAS - indicates the percentage of words for which the appropriate HEAD and LABEL were inferred.

- UAS - indicates the percentage of words for which the appropriate HEAD was inferred.
- ACC - indicates the percentage of words for which the appropriate LABEL was inferred.

The best results are highlighted column by column in bold. Our results significantly outperform FreeLing for all the four languages under studying except the LAS for English.

As expected the model with best accuracies is the one with the larger beam width (β=64) with few exceptions. But if we consider that a small increase in beam width can cause a huge increase in training and decoding time, models with smaller beam width still perform good with an accuracy drop on average between 0.5% and 1.0%.

The last rows of Table 6.3 show the accuracy scores for the two configurations using morphological features too. We observe a drop in all the scores. This is likely due to the fact that EAGLES tags already include rich morphological features and the duplication of information just cause unneeded noise.

In Chapter 5 we introduced E^t and E^l as the continuous embedded representations of all PoS tags and arc labels. Chen and Manning (2014) showed that these embeddings carry some semantic information. We wonder whether the same thing happens using our trained models. Figure 6.1 presents t-SNE visualizations of E^t embedding, the PoS tags are colored by category. It clearly shows that this embedding effectively exhibit the similarities between PoS tags.

Table 6.3: *Final dependency parsing test-set results. The β stands for beam width, the **M** indicates if morphology features are used. All the models use two hidden layers of 1024 units each, batch size of 64, learning rate of 0.07/0.02 (greedy/structure), momentum of 0.85/0.9 and a decay rate that corresponds to about one tenth of the training corpus size. All the models were trained with early stopping for a maximum of 12/20 epochs without pre-trained embeddings.*

	Catalan			Spanish			English			German		
Method	UAS	LAS	ACC	UAS	LAS	ACC	UAS	LAS	ACC	UAS	LAS	ACC
FreeLing	90.15	87.29	93.44	90.31	87.34	93.03	90.48	**89.61**	92.27	84.95	82.81	91.60
SyntaxNet β=8	91.60	88.47	93.66	91.54	88.51	93.56	91.26	88.46	92.76	88.98	86.89	94.06
SyntaxNet β=16	92.01	88.90	93.93	**92.42**	89.40	93.89	91.81	89.09	93.15	89.12	87.05	94.13
SyntaxNet β=32	92.09	89.09	94.00	92.23	89.30	93.88	91.95	89.24	93.26	89.49	87.45	94.30
SyntaxNet β=64	**92.11**	**89.14**	**94.08**	92.40	**89.50**	**93.99**	**92.07**	89.37	**93.31**	**89.73**	**87.82**	**94.57**
SyntaxNet β=16 M=1	91.96	88.78	93.78	92.32	89.04	93.41	-	-	-	-	-	-
SyntaxNet β=32 M=1	91.99	88.95	93.99	92.41	89.45	93.91	-	-	-	-	-	-

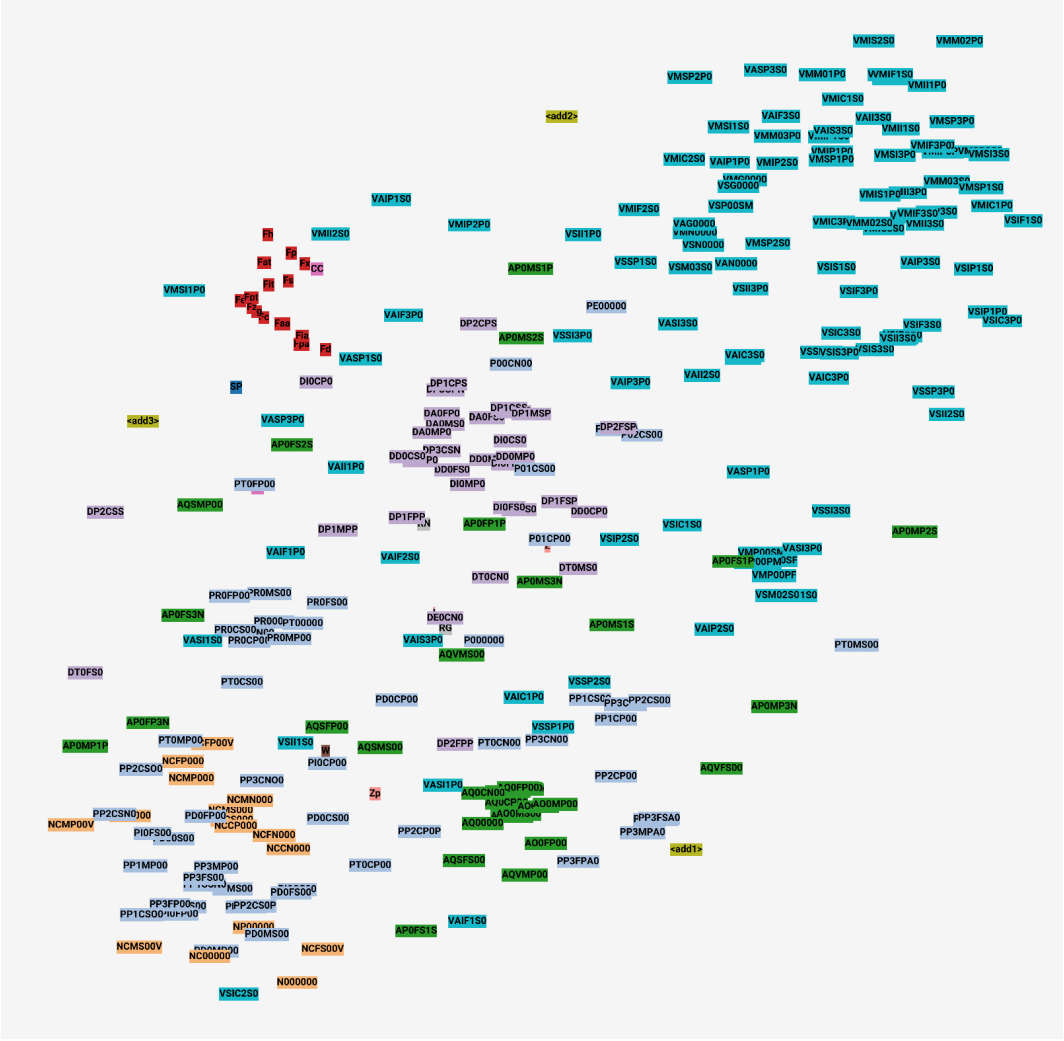


Figure 6.1: *T-SNE visualization of POS tag embeddings trained with our models. The tags are colored by categories. Tags that are similar in meaning are also closer in the embedding space.*

Chapter 7

Evaluating SyntaxNet Parameters

We performed several experiments to study how each parameter affect the behavior of the model. We present here the results. This study does not want to give precise numbers and statistics but just to show how the choice of a parameter can affect model performances.

7.1 Mini-Batch Size

The batch size defines the number of samples that are propagated through the network. When choosing the mini-batch size we need to consider:

- *time efficiency of training* - Larger mini-batches are very appealing computationally, after all considering the increased adoption of GPUs to train neural networks in the last few years.
- *noisiness of the gradient estimate* - Computing the gradient of a mini-batch corresponds in approximating the gradient of the whole training set. Given that, it's straightforward to notice that the gradient of a 100-batch is going to contain more noise than the gradient of a 100000-batch. From literature it's known that parsing effectiveness using neural networks is remarkably sensitive to the training batch size Zhou *et al.* (2015).

We performed several experiments where we kept fixed all the parameters but not the the batch size. We made sure to use the same *random seeds* to avoid that differences in performances were due to parameters initializations. On the choice of the batch size we were limited by the available computational resources. An example can be seen in Figure 7.1. From these first two examples we can notice that our models tend to perform significantly better with smaller mini-batches. Table 7.1 and Figure 7.2 confirm this results.

We think that smaller mini-batches give better accuracies because the noisiness caused by small mini-batches isn't always bad. Noisiness is regularly used in optimization meta heuristics in order to escape from local minima. Using smaller mini-batches can be seen, from an optimization point of view, as a way to get more noise in the gradient computation and hence as a way to escape a local minimum and increase the search space visited.

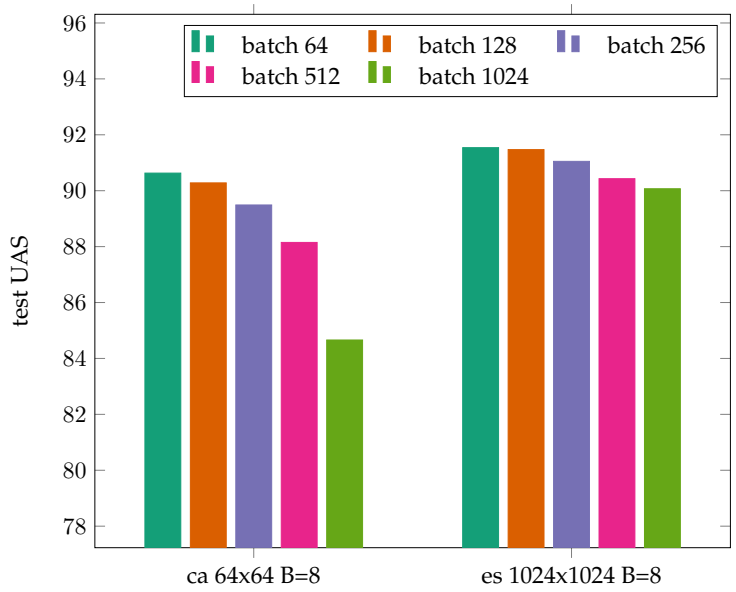


Figure 7.1: *Parsing performance with different training mini-batch sizes for two models. We used the same training epochs 12/20 (greedy/structured). As we can seen increasing the mini-batch sizes results in worse accuracies.*

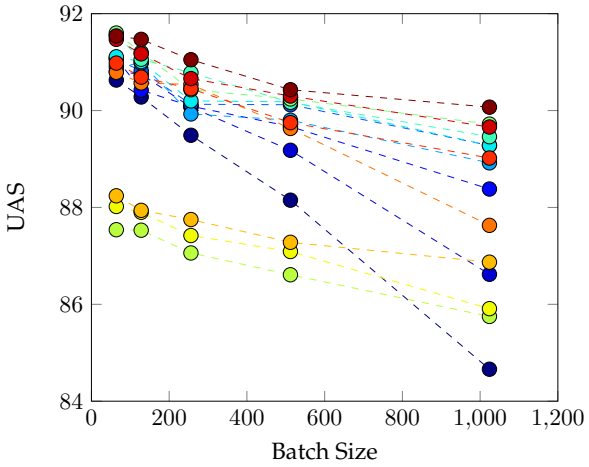


Figure 7.2: *UAS results for several configurations with different batch sizes. This uses just complete configurations where we have the results for all batch sizes.*

	Δ UAS
$64 \rightarrow 128$	-0.36 ± 0.03
$128 \rightarrow 256$	-0.61 ± 0.06
$256 \rightarrow 512$	-0.60 ± 0.07
$512 \rightarrow 1024$	-1.86 ± 0.37

Table 7.1: *Final statistics on batch size parameter. This uses all our experiments configurations no matters if they are complete or not.*

It can be argued that training with smaller mini-batches implies that the neural network parameters are updated more often and with bigger learning rates hence the higher accuracy. We tried to increase the number of learning epochs linearly with the size of the mini-batch. Figures 7.3a and 7.3b show the UAS on tuning-set for the configuration *Catalan 64x64 B=8* in the greedy training step with different batches sizes and/or training epochs. Table 7.2 contains the results for the structured training step. As we can see smaller mini-batches still outperform bigger ones: increasing the number of maximum epochs reduces the gap but

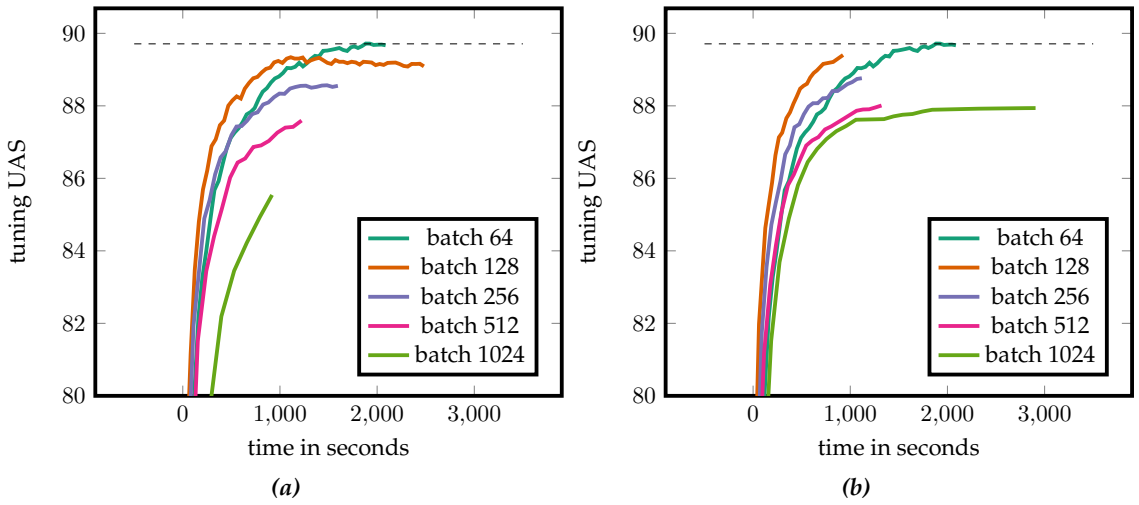


Figure 7.3: *Parsing performance with different training mini-batch sizes for the same model (**Catalan 64x64 greedy**) with same number of epochs (7.3a) or with a bigger maximum number of training epochs for larger mini-batches (early-stopping is still applied) 7.3b.*

does not fill it.

Table 7.2: *Tuning UAS results increasing the number of maximum epochs for configurations with bigger mini-batches. As we can see increasing the number of epochs helps but configurations with smaller batches still outperform the other ones.*

Hidden Layers	Beam Size	Batch Size	Epochs Greedy	Epochs Structured	tuning UAS
64x64	8	64	12	20	**90.85**
64x64	8	128	12	20	90.24
64x64			24	40	**90.28**
64x64	8	256	12	20	89.41
64x64			48	80	**89.89**
64x64	8	512	12	20	88.21
64x64			96	160	**89.94**
64x64	8	1024	12	20	84.28
64x64			192	320	**88.87**

7.2 Hidden Layers Sizes

We analyzed the parsing performances with different number of units in the two hidden layers in the Feed-Forward network of SyntaxNet (please refer to Figure 5.1). Intuitively we expected larger networks to generalize better than smaller network, nevertheless knowing that using too many neurons can result in several problems such as overfitting. Overfitting takes place when a very complex statistical model fits too closely or exactly the the observed data because it has too many parameters with respect to the number of observations. An basic example of overfitted model is shown in Figure 7.4.

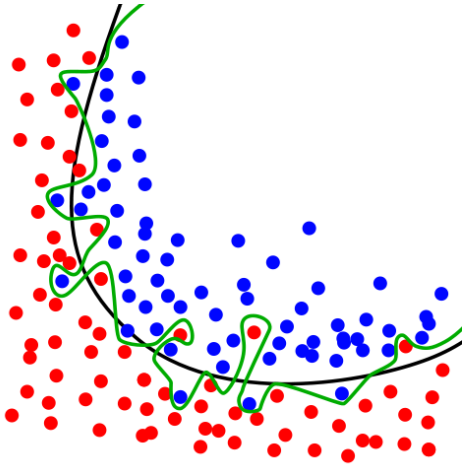


Figure 7.4: *An overfitted model (green line) vs a regularized one. The overfitted model fits too closely the training data and it will probably have a greater error rate on test data compared to the regularized one (source Wikipedia).*

Somewhat like the mini-batch experiments, we trained several models keeping fixed all the parameters but not the hidden layers sizes. Figure 7.1 shows the UAS results on test-set for three model configurations. Table 7.3 shows the final statics on all the experimented configurations. As expected larger networks tends to perform better than smaller one, but the accuracy gain is insignificant if we try to increase the sizes too much ($512 \times 512 \rightarrow 1024 \times 1024$). The risk of overfitting, measured as the difference between training UAS and test UAS, is also higher as showed in table 7.3.

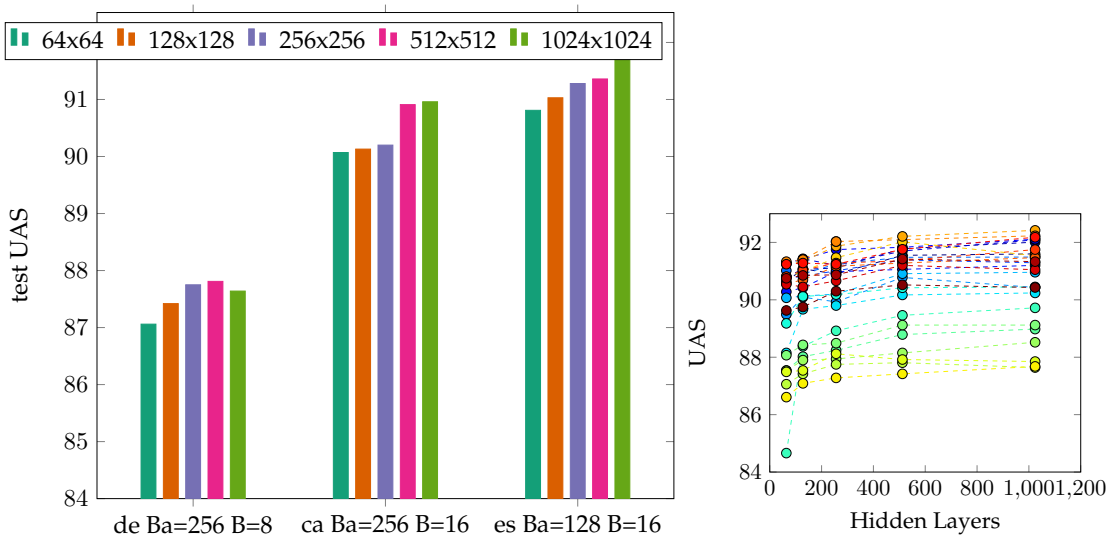


Figure 7.5: *Parsing performance with different hidden layers sizes. The left figure shows three particular configurations. The right one visualizes the trend for all the trained models.* ***Ba*** *stands for mini batch size,* ***B*** *for beam size.*

Table 7.3: *Final statistics for hidden layers sizes.*

	Δ UAS
64x64 → 128x128	0.44 ± 0.13
128x128 → 256x256	0.08 ± 0.02
256x256 → 512x512	0.20 ± 0.03
512x512 → 1024x1024	0.06 ± 0.02

	Avg Overfitting
64x64	4.00 ± 0.31
128x128	4.34 ± 0.31
256x256	4.58 ± 0.18
512x512	6.30 ± 0.25

7.3 Beam Width

Beam search is a heuristic search algorithm that reduces the number of nodes to be visited pruning at any step all non-promising nodes (Zhang, 1999). The pruning is driven by problem-specific heuristics. The group of most promising nodes is named "beam" (Xu and Fern, 2007). The size of the beam is a parameter of the search and it is usually referred to as *beam width* β. It's easy to notice that β bounds the time and memory complexity required to perform the search, at the cost of the completeness, that's to say that beam search does not guarantee that the optimal solution is found.

Same as above we trained several configurations keeping fixed all the parameters but not the beam width. Figure 7.6 shows the UAS results for three model configurations. Table 7.4 and Figure 7.7 presents the final results for all our configuration. As expected a larger beam size induces an higher accuracy.

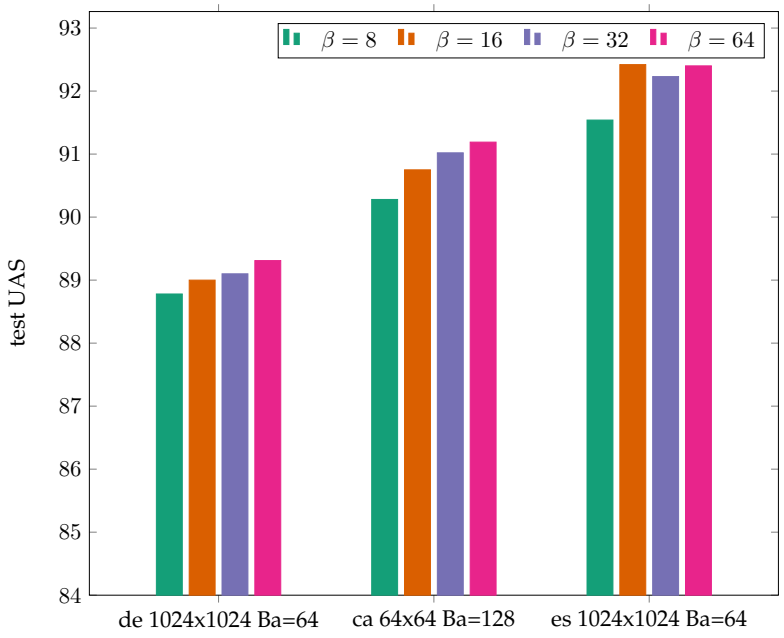


Figure 7.6: *Parsing performance with different beam widths for three models. Ba stands for mini batch size, β for beam size. Increasing the beam width results in better accuracies.*

As we can seen there is a sensible gain in accuracy increasing the beam width. Considering that even a small increment in the beam width can cause a huge amplification in the training

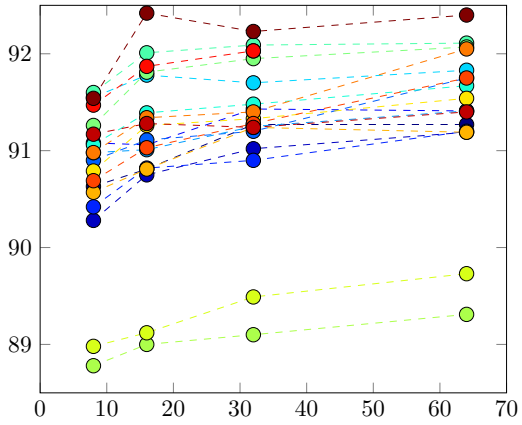


Figure 7.7: *UAS results for several configurations with different beam widths. Most of time increasing the beam width results in better accuracies.*

	Δ UAS
$8 \rightarrow 16$	0.28 ± 0.04
$16 \rightarrow 32$	0.16 ± 0.02
$32 \rightarrow 64$	0.11 ± 0.01

Table 7.4: *Final statistics for beam width parameter for all our experiments.*

and decoding times, smaller beam widths are preferable for every day use.

Chapter 8

Building a Bilingual Parser

While in transfer learning there is a sequential process, you learn from task A and then transfer to task B, in Multi-Task Learning (MTL) you kickoff concurrently trying to have one model do multiple things at the same time. MTL acts as a regularizer: forcing a model to perform well on a related task can be seen as a way to reduce overfitting. Even if MTL is successfully used in several fields, little work can be found in the literature regarding natural language.

We applied MTL to SyntaxNet. The underlying hypothesis is that some languages share a some sort of semantic structure (e.g. Catalan and Spanish, German and English, etc.). In particular we jointly trained SyntaxNet for both English and German at the same time. Limiting the study to English and German cannot make the whole picture of the situation, but we had to desist from training other languages pairs given the lack of available computational resources.

8.1 Related Works

This is not the first attempt to build a multilingual parser. Zeman and Resnik (2008) use delexicalized parsing. A parser is defined delexicalized if it is trained without any lexical features. The idea is to exploit a delexicalized treebank of resource-rich source language to train a model. The model is then used to analyze delexicalized phrases in the target resource-poor language. This take advantage of the idea that identical POS tags are highly descriptive of binary relations, and that there are shared dependency structures across languages. Petrov *et al.* (2016) and Ma and Xia (2014) take advantage of parallel corpora as a way to project constraints from the source language to the target language. Duong *et al.* (2015) proposed a model for learning a shared universal parser that builds and exploits an interlingual representation of language, in conjunction with language-specific mapping components. They also incorporate a bilingual dictionary as a set of soft constraints on the model, such that it learn similar representations for each word and its translation(s).

8.2 Model

Our main contribution is the application of the method proposed by Duong *et al.* (2015) on top of SyntaxNet. Unlike Duong *et al.* (2015) we don't use a bilingual dictionary as a set of

soft constraints on the model, but we use pretrained multilingual word embeddings. To our knowledge this if the first attempt to do such a thing.

The architecture is illustrated in Figure 8.1. We named the component introduced by Duong *et al.* (2015) as **language multiplexers**. The idea is to allow the embedding matrices to differ in order to fit language specific features. Indeed different languages can have different lexicon, part-of-speech can exhibit different roles, and dependency edges can serve different functions. In a nutshell the language multiplexer selects the language-specific embedding matrix when forwarding or back-propagating values. The multiplexer is completely optional: e.g. if training two or more languages with the same part-of-speech tags set we can use the same matrix embedding E^t for all the languages. During training, sentences are sampled from both languages to form the mini-batches.

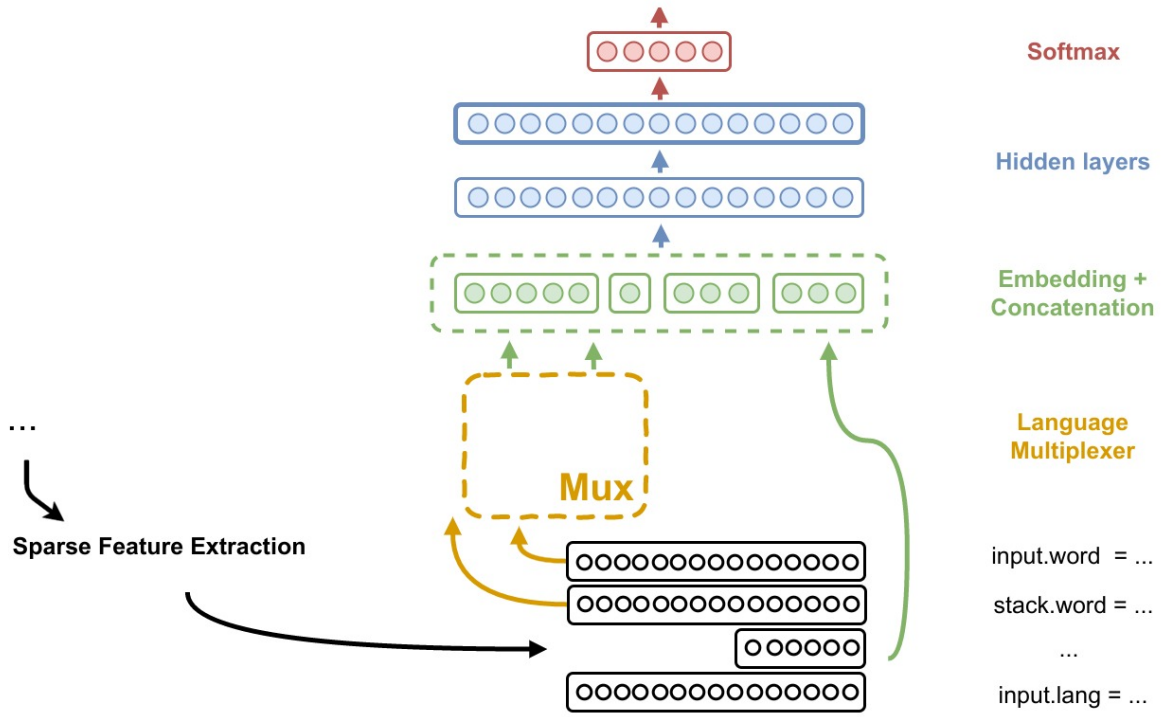


Figure 8.1: *Feed-Forward multi-lingual SyntaxNet architecture with language multiplexer.*

Please note that even if here we focus the discussion on a bilingual parser, the same idea (and the same code) can be used for multilingual parsers.

8.2.1 Bilingual Word Embeddings

As already illustrated in Chapter 4 distributed word representations have been crucial to the recent breakthrough of numerous neural network models in facing several NLP tasks. In Chapter 4 we focused on monolingual word embeddings, however the increasing interest on a wide range of multilingual NLP tasks have inspired recent work in training bilingual representations where similar words in two languages have a similar representation in the embedded space. Figure 8.2 shows an example of bilingual word embedding between German and English. Several different approaches to training bilingual word embeddings can

be used:

- Bilingual Mapping - Word embeddings are first learned for each language separately and then a (linear) transformation is found to convert embeddings from one language into embeddings of the other language.
- Monolingual Adaption - The idea is to use a well trained embeddings of a source language (e.g a resource-rich one like English), to train the target representations, making sure that embeddings of semantically related words across languages are close together.
- Bilingual Training - Word embeddings are jointly trained from the beginning.

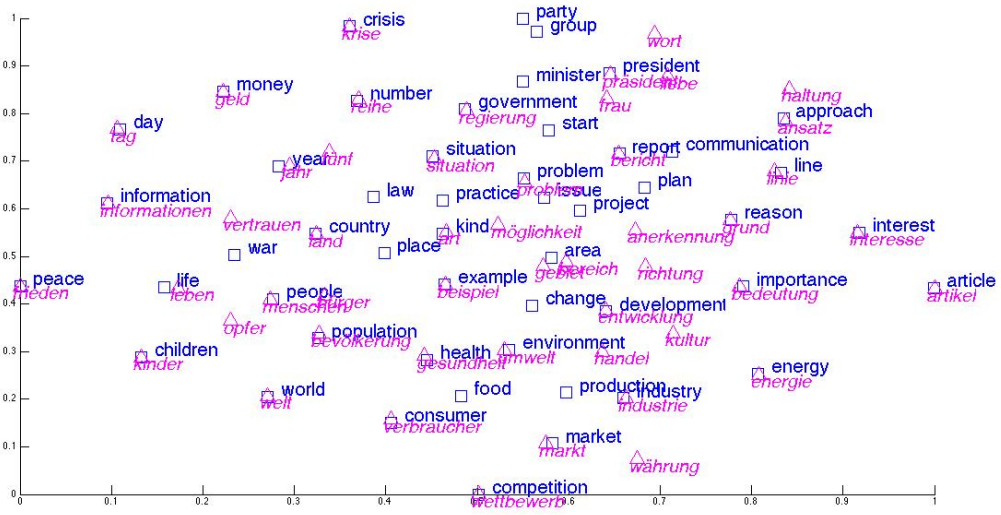


Figure 8.2: *A shared embedding space between two languages (Luong et al., 2015)*

For the purpose of this thesis we used bilingual word embeddings trained with the same recipe used by Luong *et al.* (2015). Our joint model was trained using the parallel Europarl-v7 corpus between German and English (Koehn, 2005). The corpus was pre-processed lowercasing and tokenizing the sentences and mapping each digit into 0, i.e. 2017 becomes 0000. Other uncommon terms appearing less than 5 times were mapped to `<unk>`. The resulting vocabularies are of size 95000 for German and 40000 for English . The following settings were used: SGD with a exponential decay learning rate starting from 0.025, skip-gram with context window of size 5, negative sampling with 30 samples, and a sub-sampling rate of value $1e-4$. The model is trained for 10 epochs. We set the hyper-parameters to 1 for α and 4 for β in our experiments. We used unsupervised alignment information learned by the Berkeley aligner (Liang *et al.*, 2006). For more information please refer to Luong *et al.* (2015).

8.3 Corpora

To train and evaluate the performances of our bilingual models we used the German and English corpora for the CONLL-2017 shared task [1]. Because the test sets were not yet available we had to extract them from the training sets. The corpora statistics are shown in Table 8.1.

Table 8.1: *Corpora statistics for bilingual parsing. The stats on the number of words (lowercased or non) refer just to the training set.*

	# training	# tuning	# test
English	11743	2002	800
German	13318	799	800

	# words	# lc words
English	18091	15262
German	45726	44107

8.4 Experiments

For training and decoding we used the set of features described in Table 8.2:

- Lowercased words (F^{lw}) - We used lowercased words to improve the hit rate in case we use pretrained word embeddings. In the monolingual models we used words without lowercasing them.
- Universal Tags (F^{t}) - These are the universal part-of-speech tags, shared between all the languages.
- Morphological Features (F^{m}) - It's known that morphological information proved to be valuable for parsing morphologically rich languages. These are shared between the two languages and should compensate the use of universal part-of-speech tags instead of languages specific ones.
- Labels (F^{l}) - Universal dependency relation.
- Language of the sentence under focus.

In order to reduce the training time, we performed just the greedy step with two hidden layers 1024×1024, batch size of 64, initial learning rate of 0.07, a momentum of 0.85 and a decay rate that corresponds to about one tenth of the training corpus size. All the models were trained with early stopping for a maximum of 12 epochs. We experimented both with pretrained multi-lingual word embeddings and with normal initialization.

We aim to understand if the bilingual models outperform the monolingual ones and if this can be used for low-resource languages with small or no annotated tree banks. To do so we trained our models with different ratios of the German corpus, e.g. 10%, 20%, etc with and without repetitions to guarantee a similar number of English and German sentences.

[1] http://universaldependencies.org/conll17/

Table 8.2: *Feature templates for our bilingual parser.*

	Templates
F^{lw}	i.lw i(1).lw i(2).lw i(3).lw st.lw st(1).lw st(2).lw st(3).lw st.rc(1).lw st.rc(1).ls(1).lw st.lc(1).lw st.lc(1).rs(1).lw st(1).rc(1).lw st(1).rc(1).ls(1).lw st(1).lc(1).lw st(1).lc(1).rs(1).lw st.rc(2).lw st.lc(2).lw st(1).rc(2).lw st(1).lc(2).lw
F^{t}	i.t i(1).t i(2).t i(3).t st.t st(1).t st(2).t st(3).t st.rc(1).t st.rc(1).ls(1).t st.lc(1).t st.lc(1).rs(1).t st(1).rc(1).t st(1).rc(1).ls(1).t st(1).lc(1).t st(1).lc(1).rs(1).t st.rc(2).t st.lc(2).t st(1).rc(2).t st(1).lc(2).t
F^{m}	i.m i(1).m i(2).m i(3).m st.m st(1).m st(2).m st(3).m st.rc(1).m st.rc(1).ls(1).m st.lc(1).m st.lc(1).rs(1).m st(1).rc(1).m st(1).rc(1).ls(1).m st(1).lc(1).m st(1).lc(1).rs(1).m st.rc(2).m st.lc(2).m st(1).rc(2).m st(1).lc(2).m
F^{l}	st.rc(1).l st.rc(1).ls(1).l st.lc(1).l st.lc(1).rs(1).l st(1).rc(1).l st(1).rc(1).ls(1).l st(1).lc(1).l st(1).lc(1).rs(1).l st.rc(2).l st.lc(2).l st(1).rc(2).l st(1).lc(2).l

8.5 Results

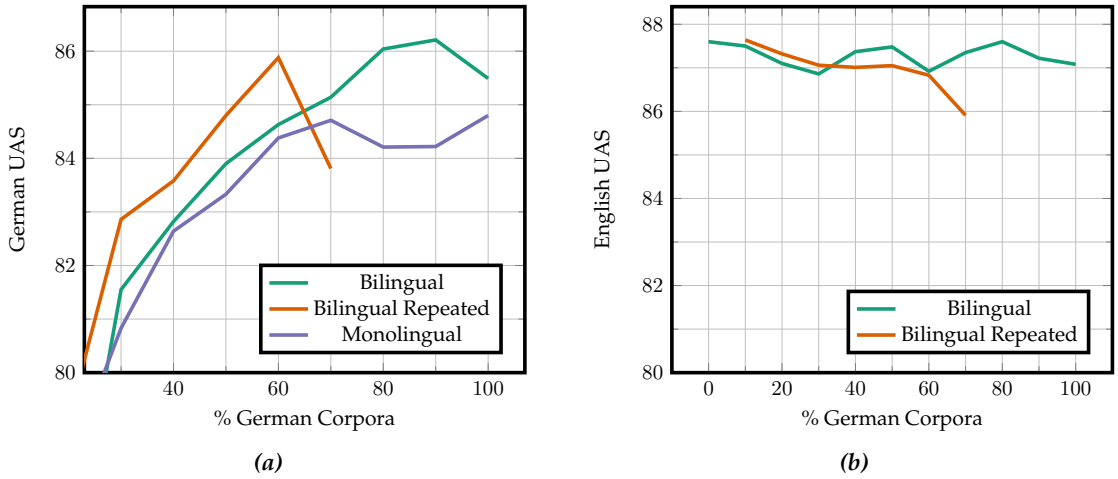
Table 8.3, Figure 8.3, and Figure 8.4 show the final results for both our monolingual and bilingual model with different German corpus ratios. As we see, at least when enforcing a similar number of German and English sentences, our bilingual model significantly outperform our monolingual models.

Even when no German corpus is used to train the model, the model gets close to 50% UAS.

Table 8.3: *English and German UAS for our monolingual and bilingual model with different German corpus ratios. Words that are similar in mean, no matter the language, are close in the embedding space.*

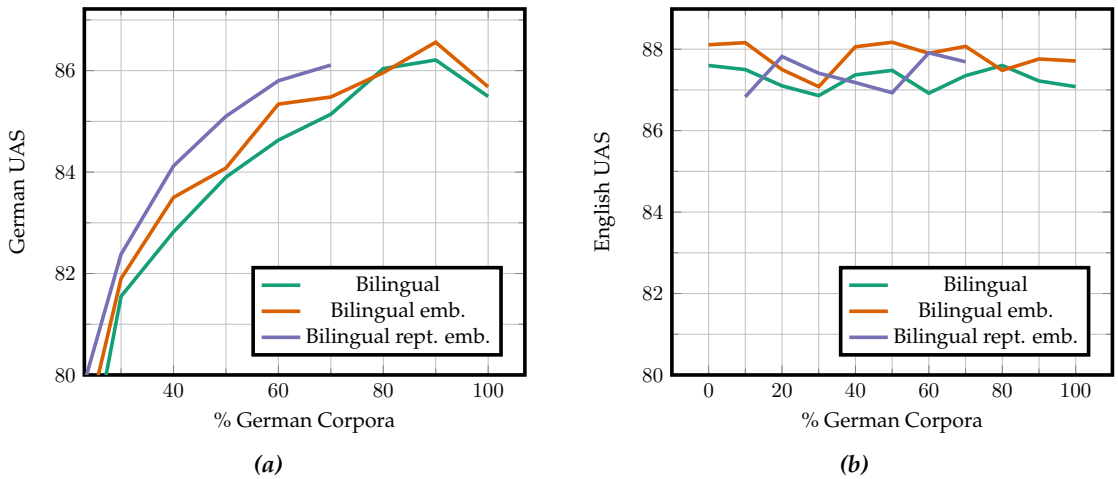
	0%		10%		20%		30%		40%		50%	
	en	de	en	de	en	de	en	de	en	de	en	de
mono	-	-	-	66.85	-	78.26	-	80.83	-	82.64	-	83.33
bi	87.6	45	87.5	68.74	87.1	76.11	86.86	81.55	87.37	82.82	87.48	83.9
bi w\rep	-	-	87.64	**72.38**	87.32	**79.11**	87.96	**82.86**	87.01	**83.58**	87.05	84.8
mono w\emb	-	-	-	66.85	-	78.13	-	80.69	-	82.70	-	83.12
bi w\emb	**88.11**	**47.68**	88.16	69.53	87.50	77.51	87.08	81.90	**88.06**	83.50	**88.17**	84.08
bi w\emb and rep	-	-	**86.83**	72.30	**87.82**	78.76	**87.41**	82.38	87.18	84.12	86.93	**85.1**

	60%		70%		80%		90%		100%	
	en	de	en	de	en	de	en	de	en	de
mono	-	84.38	-	84.71	-	84.21	-	84.22	-	84.8
bi	86.92	84.63	87.35	85.14	**87.6**	**86.4**	87.22	86.21	87.08	85.49
bi w\rep	86.83	**85.88**	85.91	83.81	-	-	-	-	-	-
mono w\emb	-	84.84	-	84.71	-	83.84	-	84.22	-	84.80
bi w\emb	87.9	85.34	**88.07**	85.48	87.48	85.96	**87.76**	**86.56**	**87.71**	**85.68**
bi w\emb and rep	**87.91**	85.8	87.69	**86.11**	-	-	-	-	-	-



(a) *(b)*

Figure 8.3: *On the left, German UAS results with our bilingual (with and without repeated training sentences) and monolingual models with different ratio of German corpus used for training the models. On the right the English UAS.*



(a) *(b)*

Figure 8.4: *Same as Figure 8.3 but using pretrained word embeddings.*

8.5.1 Effect of Corpus Repetition

We wonder if repeating sentences, to guarantee that inside a mini-batch there always a similar number of English and German sentences, help to increase the accuracy. As we can see in Table 8.5, at least for small corpus there is a substantial German UAS gain using repetition. The gain tend to decrease when the number of sentences in the German is already enough. Enforcing a similar number of English and German sentences has also a significant effect on the English UAS. This is shown in Figures 8.3 and 8.4.

8.5.2 Effect of Pre-trained Word Embeddings

. We wonder if using pretrained word embeddings help to increase the accuracy. As we can see in table 8.5, without enforcing a similar number of sentences for the two corpora, using pretrained embeddings usually comports an increase both in German and English accuracies (for the English UAS please refer to table 8.3). When enforcing a similar number of sentences

Table 8.4: *Effect of enforcing a similar number of English and German sentences for different ration of the German corpus, with and without pretrained word embeddings. The reported accuracy is the German UAS.*

	10%	20%	30%	40%	50%	60%	70%
bilingual	68.74	76.11	81.55	82.82	83.9	84.63	**85.14**
bilingual w\rep	**72.38**	**79.11**	**82.86**	**83.58**	**84.8**	**85.88**	83.81

	10%	20%	30%	40%	50%	60%	70%
bi w\emb	69.53	77.51	81.90	83.50	84.08	85.34	85.48
bilingual w\emb and rep	**72.30**	**78.76**	**82.38**	**84.12**	**85.1**	**85.80**	**86.11**

for the two corpora, this is not always true.

Table 8.5: *Effect of using pretrained word embeddings, with and without corpus repetition. The reported accuracy is the German UAS.*

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
bilingual	45	68.74	76.11	81.55	82.82	83.9	84.63	85.14	**86.4**	86.21	85.49
bilingual w\emb	**47.68**	**69.53**	**77.51**	**81.90**	**83.50**	**84.08**	**85.34**	**85.48**	85.96	**86.56**	**85.68**

	10%	20%	30%	40%	50%	60%	70%
bilingual w\rep	**72.38**	**79.11**	**82.86**	83.58	84.08	**85.88**	83.81
bilingual w\rep and emb	72.30	78.76	82.38	**84.12**	**85.10**	85.80	**86.11**

Chapter 9

Conclusion

We introduced the reader to SyntaxNet going through the concepts of dependency parsing, transition-based dependency parsing and neural network models for transition-based dependency parsing. We then trained SyntaxNet for four different languages and compared the accuracies with FreeLing, showing a significant gain in accuracies. We then experimented with several configurations, tweaking the mini-batch size, the number of units in the hidden layers and the beam-width. Somehow unexpected we showed that small mini-batches tend to perform better even if similar studies on similar networks showed the exact opposite. This has big implications for everyone training a SyntaxNet model: training with larger mini-batches is preferable to reduce the training times but will deeply affect the accuracies of the system.

Last but not least we built a bilingual parser, training jointly a model for English and German. The idea is that we can exploit similarities between languages. We performed similar experiments with different ratios of the German corpus, in order to understand if our bilingual parser can be used to improve dependency parsing model for languages with small or no-existing tree banks. We experimented using bilingual word embeddings and forcing a similar number of English and German sentences inside a mini-batch, showing that both techniques significantly help to increase the accuracies of the trained models. The results on the multi-lingual parser can be extremely useful in several applications. In all the cases we have to deal with low-resource languages a multi-lingual model can be effectively used to get competitive results. Considering also the increasing interest in performing inference directly in embedded devices instead that in powerful servers, this work can be used to build models that can deal with more than one language and at the same time requiring a limited amount of memory and storage resources.

9.1 Future Work

For future work we aim to understand how bilingual models help with languages with even bigger similarities, e.g. Catalan and Spanish, and with languages with low similarities. It's also interesting to understand how the model behave when training with three or more languages.

We also acknowledge the limit of the used pretrained word embeddings. Some recent works explored the idea of using Gaussian (or mixture of Gaussians) word embeddings.

Those have some nice properties:

- they can deal with words with more that one meaning,
- they inherently have a hierarchic structure, e.g. in standard word embeddings the word *fruit* and *apple* are close together in the embedded space, but there is no way to represent (and to learn) that *apple is a fruit*.

In other to go further with Gaussian word embeddings several issues needs to be addressed, that to our knowledge are still open or not very well understood:

- using Gaussian word embeddings would imply dealing with neural networks accepting uncertainty in the input and/or in the hidden layers.
- how to train multilingual Gaussians word embeddings?

Considering that multilingual parsers, and more in general multilingual models for NLP tasks, can be seriously useful in the embedded word, further study are required comparing and/or combining our proposed technique with other ones, e.g. quantization.

Last but not least our bilingual parsers improves accuracy on target languages but proved to sacrifice accuracy on source languages. Further studies are required to study and possibly stem this phenomenon.

Bibliography

Erick Alphonse, Sophie Aubin, Philippe Bessières, Gilles Bisson, Thierry Hamon, Sandrine Lagarrigue, Adeline Nazarenko, Alain-Pierre Manine, Claire Nédellec, Mohamed Ould Abdel Vetah, Thierry Poibeau, and Davy Weissenbacher. 2004. Event-based information extraction for the biomedical domain: The caderige project. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and Its Applications*, JNLPBA '04, pages 43–49, Stroudsburg, PA, USA. Association for Computational Linguistics.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Gosse Bouma, Jori Mur, Gertjan van Noord, Lonneke van der Plas, and Jörg Tiedemann. 2006. *Question Answering for Dutch Using Dependency Relations*, pages 370–379. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-45700-8.

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

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Simon Corston-Oliver Chris Quirk. 2006. The impact of parse quality on syntactically-informed statistical machine translation. In *Proceedings of EMNLP 2006*. ACL/SIGPARSE.

Fabio Ciravegna and Alberto Lavelli. 1999. Full text parsing using cascades of rules: An information extraction perspective. In *Proceedings of the Ninth Conference on European Chapter of the Association for Computational Linguistics*, EACL '99, pages 102–109, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael Collins. 2002. 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.

Hang Cui, Keya Li, Renxu Sun, Tat seng Chua, and Min yen Kan. 2005a. National university of singapore at the trec 13 question answering main task. In *In Proceedings of the 13 th TREC*.

Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. 2005b. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 400–407, New York, NY, USA. ACM. ISBN 1-59593-034-5.

Yuan Ding and Martha Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 541–548, Stroudsburg, PA, USA. Association for Computational Linguistics.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. A neural network model for low-resource universal dependency parsing. In *EMNLP*, pages 339–348. Citeseer.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Comput. Linguist.*, 39(4):799–845. ISSN 0891-2017.

Valentin Jijkoun, Maarten De Rijke, and Jori Mur. 2004. Information extraction for question answering: Improving recall through syntactic patterns. In *In Coling 2004*, pages 1284–1290.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 17–24, Stroudsburg, PA, USA. Association for Computational Linguistics.

Percy Liang, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 104–111. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Bilingual word representations with monolingual quality in mind. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 151–159.

Xuezhe Ma and Fei Xia. 2014. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In *ACL (1)*, pages 1337–1348.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330. ISSN 0891-2017.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4):513–553. ISSN 0891-2017.

Lluís Padró and Evgeny Stanilovsky. 2012. Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*, Istanbul, Turkey, May 2012. ELRA.

S. Petrov, R. McDonald, and K. Hall. 2016. Multi-source transfer of delexicalized dependency parsers. US Patent 9,305,544.

R. Sætre, K. Sagae, and J. Tsujii. 2008. Syntactic features for protein-protein interaction extraction. In C. J. O. Baker and J. Su, editors, *Short Paper Proceedings of the 2nd International Symposium on Languages in Biology and Medicine (LBM 2007)*, volume ISSN 1613-0073319, pages 6.1–6.14, Singapore, January 2008. CEUR Workshop Proceedings (CEUR-WS.org).

Libin Shen, Jinxi Xu, and Ralph Weischedel. 2008. A new string-to-dependency machine translation algorithm with a target dependency language model. In *In Proc. of ACL*, pages 577–585.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*.

Yuehua Xu and Alan Fern. 2007. On learning linear ranking functions for beam search. In *Proceedings of the 24th international conference on Machine learning*, pages 1047–1054. ACM.

Daniel Zeman and Philip Resnik. 2008. Cross-language parser adaptation between related languages. In *IJCNLP*, pages 35–42.

Weixiong Zhang. 1999. *State-space search: Algorithms, complexity, extensions, and applications*. Springer Science & Business Media.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *ACL (1)*, pages 1213–1222.