Parsing morphologically-rich languages using neural networks

Andreas Grivas

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2017

Abstract

In this dissertation we explore ways of adapting dependency parsing for languages with rich morphological features using neural networks. Our work leverages recently proposed neural network subword representations (Kim et al., 2015; Ling et al., 2015) that theoretically have the potential of capturing morphological patterns. Conceptually, we follow the work of Vania and Lopez (2017) who compare such representations on a variety of languages for the task of language modelling. We port some of the encoders which they report work well to dependency parsing and obtain competitive results for 3 languages on the CONLL 2006 (Buchholz and Marsi, 2006) dataset and for 7 languages on the Universal Dependencies (Nivre et al., 2016) v1.3 dataset. Our results highlight that the subword representations of Ling et al. and Kim et al. provide substantial improvements when used for parsing most morphologically-rich languages compared to word level models. They suggest that the improvements come due to their capability of mitigating the out of vocabulary issues word models have. While we found evidence that our subword model might be capturing case inflection in a number of examples we demonstrate, it is still not clear whether in general these subword models capture salient morphological information useful for discerning the syntactic structure of the sentence. Given that we observe a sharp drop in performance when part of speech tags are not used as input to our models, we conclude that guiding the subword representations with additional information about the input is still a vital resource for dependency parsing datasets of this size.

Acknowledgements

I would like to thank my supervisors, Adam Lopez and Clara Vania, for putting together this exciting project, their guidance and extensive insightful feedback. Sameer Bansal for discussing neural architectures, Marco Damonte, Federico Fancellu, Sorcha Gilroy, Yova Kementchedjhieva, Junyi Li, Joana Ribeiro, Ida Szubert, Ieva Vasiljeva and Ajay Viswanathan for feedback on drafts and sharing interesting papers throughout the dissertation period.

My colleagues Marcin Müller, and Jeff Ward, for fruitful discussions and insights throughout the semester. Stephen Graham with his witty puns and ideas, Aagje van der Meer, Fernando Paulin, Ivan Ho, Luke, Laura and Panagiotis Efstratiadis for making this intense year have less effect on my sanity.

My parents, for their patience and immense support in all shapes and forms. My Grandma Stella Griva and my Grandad Dennis Turner for helping me fund this Masters, making it possible for me.

Finally, I'd like to thank Marina for putting up with my bad humour, for adding `<span style="color: #0F0">colour</span>` to this long year and for being my front end developer.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Andreas Grivas*)

Table of Contents

List of Figures

List of Tables

Nomenclature

$\langle A, B \rangle$ Frobenius inner product: $\langle A, B \rangle = Tr(\boldsymbol{A}\boldsymbol{B}^T)$

$\odot$ Hadamard product: $[x_1, x_2, \ldots x_n] \odot [y_1, y_2, \ldots y_n] = [x_1 y_1, x_2 y_2, \ldots x_n y_n]$

$\sigma(x)$ Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

$e_w(x)$ Embedding lookup of x in $\boldsymbol{E}_w$: $e_w(x) = \boldsymbol{E}_w x_{onehot}$

BILSTM Bi-directional Long Short Term Memory network

CNN Convolutional Neural Network

LAS Labelled Attachment Score

LSTM Long Short Term Memory network

OOV Out Of Vocabulary

UAS Unlabelled Attachment Score

A centipede was happy quite!
Until a toad in fun
Said, "Pray, which leg moves after which?"
This raised her doubts to such a pitch,
She fell exhausted in the ditch
Not knowing how to run.

Katherine Craster

1 Introduction

In this dissertation we explore ways of adapting dependency parsing models for languages which tend to express syntactic and semantic information through modifications of their word forms. Such dependency parsing models are used to obtain the syntactic relationships between words in a sentence, providing insight about its meaning. An example of what such an analysis for an English sentence would look like is illustrated [1] in Figure 1.1.

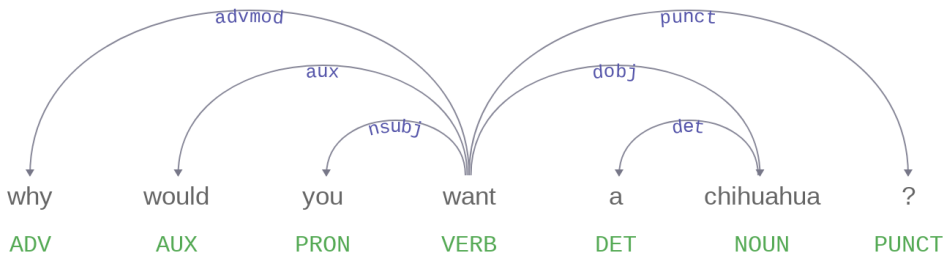


Figure 1.1: Dependency parse for a sentence in English; a morphologically-poor language. From this parse we can determine that "you" is the nominal subject (nsubj) of the verb "want", while "chihuahua" is its direct object (dobj). The green labels below the words denote the part of speech tags. We shall delve into the specifics of dependency parses in Section 2.2.

[1] Credits for dependency graph visualisations go to https://demos.explosion.ai/displacy/.

Our objective is to create models that generate such output, but do so effectively for morphologically-rich languages. Such languages which include Finnish, Czech and Ancient Greek to name but a few, have a high type to token ratio; the number of distinct word forms in a text is inflated due to the increased number of morphological markers that can augment the base form of each word.

Switching from chihuahuas to foxes and from morphologically-poor to morphologically-rich, here is an example in Ancient Greek from Aesop's fables.

(1.1)

*T*οῦτο	ἀλώπηξ	κατιδοῦσα	ἐβόα	.
acc neut sing	nom fem sing	nom fem sing past act	3^{rd} sing past act	.
This	fox	saw	cried aloud	.

"When the fox saw this it cried aloud ."

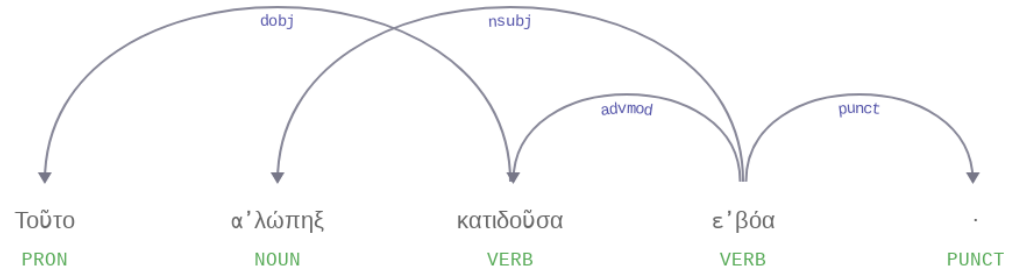


Figure 1.2: Dependency parse for an Ancient Greek sentence from Aesop's fables. The fox ("ἀλώπηξ") is the subject of cried aloud ("ἐβόα"), while this ("Τοῦτο") is the object of saw ("κατιδοῦσα").

In Ancient Greek the word forms of verbs and nouns are modified to express morphological information such as tense and case respectively. More interestingly, "κατιδοῦσα"—a participle by the books—is a hybrid of a verb and a noun; a form of gerundy participle. For the purposes of our current discussion, suffice it to say that such forms in Ancient Greek express characteristics of both nouns and verbs. Given the distinct number of choices for case, gender, number, tense and voice, such participles can have up to $5 \times 3 \times 2 \times 4 \times 2 = 240$ distinct [2] word forms, some examples of which are tabulated in Table 1.1. To contrast this to morphologically-poor English, *-ing* participle forms, such as "crying", do not change when we want to express a different tense. Instead, we express the tense by inflecting the accompanying auxiliary verb, as exemplified in the following examples.

[2] We ignore the effects of case syncretism.

Modification	word form
—	ἡ κατιδοῦσα
voice=passive	ἡ κατιδομένη
tense=future	ἡ κατοψομένη
case=dative	τῇ κατιδούσῃ
case=accusative, gender=male	τόν κατιδόντα
tense=present, gender=neuter	τό καθορῶν
tense=present, gender=neuter, number=plural	τά καθορῶντα

Table 1.1: Example of how the word form for Ancient Greek participle "κατιδοῦσα" changes depending on expressed case, gender, number, tense and voice. We express all modifications based on "κατιδοῦσα" which is nominative, female, singular, past and passive respectively. Its determiner which also expresses the same case, gender and number is also prepended.

(1.2) The fox *is* crying (present)
The fox *was* crying (past)
The fox *has been* crying (past perfect)

Similar vocabulary inflating effects to those demonstrated for Ancient Greek are also present in other morphologically-rich languages. Czech, for example, has 7 cases for nouns which is more than the 5 which Ancient Greek possesses. The overwhelming number of possible word forms in these languages alone make them much more challenging to parse. This is so since a large number of possible inflected word forms—especially those of rare words—will not be present in our dataset regardless of its size. Moreover, even if the word forms were present, they would quickly saturate any finite sized vocabulary, rendering this approach infeasible without resorting to the highly adopted but inelegant solution of modelling some words as "unknown". As should be clear, employing word models to parse morphologically-rich languages in such a setting, suffers from a high out of vocabulary rate (many unknown words) when processing novel texts not in our dataset. On a final note, even if all the rare words could be represented in the vocabulary, machine learning methods underperform when generating representations given scarce examples.

Unsurprisingly, the reported performance of dependency parsers that operate with a word vocabulary on morphologically-rich languages is much worse than for English (Buchholz and Marsi, 2006; Nilsson et al., 2007; Tsarfaty et al., 2010). To justify why this is so, we urge the reader to consider the comparison of inflected participles with respect to tense we went through previously. For Ancient Greek, each form of the participle will result in a new word form the dependency parsing model will have to learn a representation for. On the other hand, for English it need only learn representations for the base form and the inflections of the accompanying verbs. Crucially, English verbs have very few inflections and the accompanying auxiliary verbs are extremely common. To highlight the differences in morphological complexity between English and other morphologically-rich languages even further, Cotterell and Schütze (2015) report that for their dataset Czech had 970 possible morphological tags while English had 137.

The aforementioned disparity in performance makes dependency parsing for morphologically-rich languages an exciting problem to work on, since there are non-trivial potential gains to be obtained if existing methods are improved. Moreover, the potential gains are multiplicative; there is a vast number of morphologically-rich languages, and for each language, models that can access morphological information have the potential to outperform morphologically unaware ones by a significant margin.

Given this incentive, we explore dependency parsing models that leverage recently proposed neural network subword representations (Ling et al., 2015; Kim et al., 2015). These models mitigate the out of vocabulary issue by constructing representations for words compositionally from characters. The former does so by employing a recurrent neural network, the latter a convolutional neural network. These approaches work well because the character vocabulary is guaranteed to be orders of magnitude smaller than the corresponding word vocabulary.

As put by Kim et al., the previously introduced subword models are character aware; they theoretically have the potential of capturing morphological information. Despite their successful use on various tasks and languages however, it is still unclear what patterns these models capture. **Are subword models actually capturing morphological information that would be useful in our case for discerning syntactic structure?**

In order to investigate what the subword models are doing differently, we follow the work of Vania and Lopez (2017) but for dependency parsing; we compare

the previously introduced subword models to word models on morphologically-rich languages such as Czech, Finnish and Ancient Greek. We seed these subword architectures with the settings they report worked well, but modify the sizes of the networks such that all models have the same number of trainable parameters. This is done to compare the models on an equal footing and to narrow down the search space of hyperparameters. We then fine-tune the remaining hyperparameters for each model by taking into account performance across representative languages and compare the best setup for each model on two datasets. The research questions we wish to elucidate by making this comparison are as follows:

- **To what extent do recent neural network based models proposed for subword level encoding of language capture morphological information?**
- Given that information about the structure of word forms can provide clues about the relationships between the words in the sentence(Lee et al., 2011), do such representations improve parsing performance for languages which are rich in such information?
- How do such subword representations compare to word level ones, which cannot by construction capture subword information and therefore be aware of morphology?
- Do subword models make morphological analysis redundant or do they merely come up with a more effective representation for words?

We note that we will provide evidence to support our claim regarding the last question posed, however a more thorough analysis is in order, as we shall discuss in Section 5.

At this point we summarise what we believe to be the contributions of this dissertation before providing a brief outline of the following sections. The contributions of this dissertation are the following.

1. We introduce a graph dependency parser, a hybrid of Zhang et al. and Kiperwasser and Goldberg. We extend its encoder to construct word representations compositionally from characters using recurrent neural networks (Ling et al., 2015) or convolutional neural networks (Kim et al., 2015). Using these subword encoders we obtain competitive results on three languages on the

CONLL 2006 (Buchholz and Marsi, 2006) dataset and seven languages on the Universal Dependencies (Nivre et al., 2016) dataset.

2. We find that both Ling et al. and Kim et al. subword models work well for most morphologically-rich languages with minor differences between the two. This is in contrast to what we found reported in the literature, which seems to be in favour of the convolutional network character encoder (Chorowski et al., 2016; Yu and Vu, 2017; Ma and Hovy, 2017).

3. We provide clear evidence that subword models help mitigate the out of vocabulary issue by learning alternative, more effective representations for words. However, we find that these models still depend a lot on linguistic input to learn good representations; we demonstrate that the char-BILSTM model suffers sharp drops in parsing accuracy when no part of speech input is used.

4. By carrying out a qualitative analysis on cherry-picked Ancient Greek[3] sentences, we find that the differences between the word model and the character model are clear, but not necessarily predictable. More importantly, we find evidence that the subword model is capturing case and agreement phenomena much better than the word level model. However, the extent to which it does this and whether the model generalises to a greater number of examples is a matter for future work.

The outline of the rest of the dissertation is as follows.

In Section 2, we provide the background for our discussion. We begin by providing some background material on morphology. Then, we introduce dependency parsing and position our approach with respect to related work.

Next, in Section 3 we illustrate the three models we implemented, and motivate our choice of hyperparameters for them.

In Section 4, we report results for our three models on the CONLL 2006 and Universal Dependencies v1.3 datasets. We then carry out a qualitative analysis on Ancient Greek sentences to probe the inner state of our word model and our recurrent character model.

To conclude, in Section 5 we summarise our findings and point in directions for future work.

[3]Ancient Greek was chosen for the examples because due to the similarities to Modern Greek, the author was more competent with it than other morphologically-rich languages.

2

Background

In this section we provide the main concepts needed to make the experiments and results in the forthcoming sections intelligible. We first briefly explain what morphological processes we will be interested in and describe a conventional way of classifying languages into groups known as morphological typologies. In the second part, we illustrate dependency parsing, position our approach amongst the relevant literature and introduce the datasets and evaluation metrics used.

2.1 Morphology

2.1.1 Definitions & key concepts

Morphology is concerned with the internal structure of words and the processes by which they are formed (Bauer, 2003). Words are made up of smaller segments called **morphs**. These are considered to be realisations (surface forms) of abstract semantic units, called **morphemes**. The examples in this section and the next are from Bauer (2003).

Words are produced by bringing together base forms and affixes. An affix depending on how it is joined to a base can, among other more rare cases, be:

- a prefix: $create \rightarrow re \cdot create$ (English)
- a suffix: $create \rightarrow create \cdot s$ (English)
- a circumfix: $film \rightarrow ge \cdot film \cdot t$ (German)
- a transfix: $ktb \rightarrow katab$ (Egyptian Arabic)

Affixes are also classified according to the result they produce.

Derivational affixes produce new words that may change the part of speech. They are usually also distinct semantically from the original word.

(2.1) $re \cdot create$

Inflectional affixes produce new surface forms of a given word. They don't change its part of speech, they annotate the word with additional syntactic information.

(2.2) $create \cdot s$

We note, that for the purposes of this dissertation we shall not be differentiating between these two different forms of word production. Languages generally vary on the degree to which they use word order and morphological markers to highlight syntactic relationships. In the next subsection we introduce a coarse classification of languages into morphological typologies.

2.1.2 Morphological typologies

No coarse classification such as the following can be accurate, most languages share characteristics from many different groups. We will give an overview of such classifications here though to get a sense of the diversity of languages that exist. Languages are classified into isolating, agglutinative and fusional [1].

Isolating languages have a very low morpheme-per-word ratio and have no inflectional morphology. An example of such a language is Chinese.

In **agglutinative** languages, each morph can be more easily attributed to a single morpheme, making the individual morphs easier to tease apart. As an example from Finnish, $talo \cdot i \cdot ssa \cdot an$.

[1] some also add polysynthetic, but we don't have such a language in a current dataset.

(2.3) *talo* *i* *ssa* *an*
house plural in 3^{rd}person-possessive
"in their houses"

In **fusional** languages, a particular affix may express many morphemes. As an example from Latin, in the word $ann \cdot um$, um expresses both the accusative case and the singular number.

(2.4) *ann* *um*
year accusative+singular
"year"

Many disagree in general with this classification of languages into typologies. Languages change constantly. As an example of drift—from rigid inside the word to rigid in syntax—here is an example sentence in Ancient Greek and Modern Greek. Note that in Ancient Greek the past perfect of the verb write "γέγραφα" is a single word. In Modern Greek it is formed using an auxiliary verb, exactly like in English.

(2.5) ἀπεκρίθη ὁ *Π*ιλᾶτος *Ὃ* γέγραφα, γέγραφα.
responded Pilate that which I have written, I have written.
'Pilate responded: that which I have written, I have written.'

(2.6) *Α*ποκρίθηκε *ο Π*ιλάτος*:* ό,τι έχω γράψει, έχω γράψει.
responded Pilate: whatever I have written, I have written.
'Pilate responded: whatever I have written, I have written.'

2.1.3 Morphologically-rich languages

Cotterell and Schütze (2015) define morphologically-poor languages to be:

"Languages with a low morpheme-per-word ratio such as English".

Since we are focussing on morphologically-rich languages, we shall reverse their definition to obtain:

Definition 1. *A language is morphologically-rich if it has a high morpheme-per-word ratio.*

This is a sound definition, however it is not very practical. We do not have information about morphemes readily available in our dataset. We could use a morphological analyser to gather such information, but usually these systems are language specific and far from perfect. As a compromise, we will use the type

to token ratio and out of vocabulary percentage as a proxy for signaling how morphologically-rich a language is.

Three issues we need to be aware of when dealing with morphologically-rich languages are the following.

1. as argued in introduction the introduction, we need to deal with the high out of vocabulary rate.

2. morphologically-rich languages can have free word order, hence we need to deal with non-projectivity (we will introduce this concept in Section 2.2.2.3)

3. morphology is not independent of syntax, as Lee et al. (2011) claim, it is a chicken and egg problem and needs to be dealt with jointly Tsarfaty et al. (2010).

Another challenge of morphologically-rich languages is that sometimes they drop the subject or object of the sentence if it can be inferred from the the rest of the sentence. Such languages which include Greek, Italian and Japanese among others are typically referred to as pro-drop (pronoun-drop). As an example, in Modern Greek it is common to drop the subject of a sentence if is clear from the context.

(2.7) *Π*οιός είναι ο *Γ*ιώργος*;*
Who is George?
"Who is George?"

(2.8) *E*ίναι ο δήμαρχος
Is the mayor
"He is the mayor"

Another interesting observation is that modelling Ancient Greek on the sub-word level is compelling for an additional reason. The hypogegrammenh is an overt marker for the dative case.

(2.9) ἔγνων, Ὀδυσσεῦ, καὶ πάλαι φύλαξ ἔβην τῇ σῇ
I know it, Odysseus, and long ago guardian I came thy
πρόθυμος εἰς ὁδὸν κυναγίᾳ.
willing at road hunt.
'I know it, Odysseus, and some time ago I came on the path as thy guardian friendly to your hunt'

As can be seen from the example, being able to model the hypogegrammenh would help model dependencies between τῇ, σῇ and κυναγίᾳ, since they agree in case. We believe that such characteristics of the written form of Ancient Greek is a significant contributing factor to the difference we observe in performance between word level models and subword level models as we shall see in 4. As we shall describe in Section 3.2.1.2, we split diacritics of characters to form separate tokens such that the model can even more effectively capture them.

2.2 Dependency Parsing

In this section, we begin by introducing dependency parsing. We shall then analyse the structure of a dependency graph more formally and define some of their important properties. Finally, we shall explain how our parser relates to the literature.

2.2.1 Introduction and historical context

This subsection sets the historical background and is mainly introductory. While we believe it contains an interesting discussion comparing dependency parsing theory with current practice, it doesn't have any dependencies—only stemmas—and can thus be safely skipped.

Parsing can be thought of as the procedure of building a syntactic structure over a sequence of words, capturing which words relate to which others in the sentence. An intuitive way of introducing syntactic structure, as carried out in Tesnière (2015)[2], is by noting that language is observed as a "spoken chain"; it is constrained to be a sequence of utterances in a single dimension. Under such a constraint, each utterance can only have an explicit spatial relation to its two neighbours. Such a representation would be insufficient for weaving the complex relationships between utterances humans can express using language. We therefore assume that we have a higher dimensional structured representation for language than what is observable. Under this assumption, parsing is the procedure of reconstructing the relationships between the words that were lost when projecting this structured representation into a stream of utterances.

Tesnière is considered to be the founding father of dependency grammar.

[2]We cite the translation, the book is in French: Les Éléments de syntaxe structurale (1959)

We shall discuss some of his work here as a brief introduction, but also in order to highlight some interesting key differences with current dependency parsing format (Nivre et al., 2016) we will be using, pointing out some of its limitations.

He applied his theory to an extensive variety of languages, including French, Russian, Polish, Ancient Greek and Latin. This fact explains why dependency grammars have been found to be flexible in describing a variety of languages, even those that do not improse a rigid order constaint on their words. He illustrated his syntactic analysis employing constructs he referred to as "stemmas". In Figure 2.1 we provide an example of such a syntactic representation for a French sentence. We note that the syntactic analysis has a single root, in this case the verb "est", which all other words are subordinate to. Key differences from the dependency graph formalism we will introduce shortly are the following:

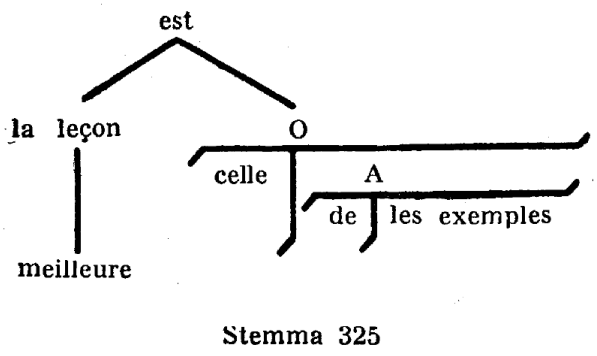


Figure 2.1: Stemma (parse) from Tesnière's Elements for "la meilleure leçon est celle des examples".

- The stemma does not take into account the original order of the words in the sentence. In contrast, dependency graphs are grounded in the sentence and can be read from left to right[3].
- The nodes in a stemma are more complex than the equivalent nodes in dependency graphs. They some times comprise of more than one word; a full word that can stand on its own and optionally more functional words that modify the full word to form the node as a whole.
- Tesnière only identified 4 parts of speech (Noun, Adjective, Verb and Adverb) for full words. He advocated that all other types of functional words,

[3]Assuming the parse is not for Arabic!

such as prepositions, act compositionally on full words to form a larger "nucleus" potentially with a modified part of speech. This composition is what the tau shaped construction represents in his stemmas, a notion he named transfer (translation).

- While the arcs in the stemmas are directed (even though the direction is not explicitly marked), Tesnière represents conjunctions using a different type of undirected arc. Conversely, in dependency graphs all arcs are directed and no distinction is made between them.

- Stemmas can include words as nodes that aren't present in the sentence, such as dropped subjects.

To elaborate on the last point, as we saw in the previous section, some morphologically-rich languages tend to drop the subject of the verb when it is easily understandable given the morphological markings of the verb and the context of the sentence. Interestingly, Tesnière (2015) accounted for such cases in his syntactic analysis. This can be seen in examples from Ancient Greek and Latin in Figure 2.2, where he represents the dropped subject as (nom). As we shall see in Section 2.2.2.3, our approaches to dependency parsing fall short in such cases; they cannot draw arcs to words that are not explicitly present in the sentence.

(2.10) ὁμολογῶ εἶναι Ἀθηναῖος
Admit be Athenian
"I admit that I am Athenian"

2.2.2 Definitions & key concepts

We shall now define dependency graphs in a mathematical manner, following Kübler et al. (2009). We do so mainly to explain the concept of projectivity, which is in turn needed to comprehend the difference between our decoding algorithms.

As we have already alluded to, dependency graphs encode the relationships between the words in a sentence. These relationships are considered to be binary and asymmetric, and can be visualised as directed arcs connecting a governing word to a subordinate.

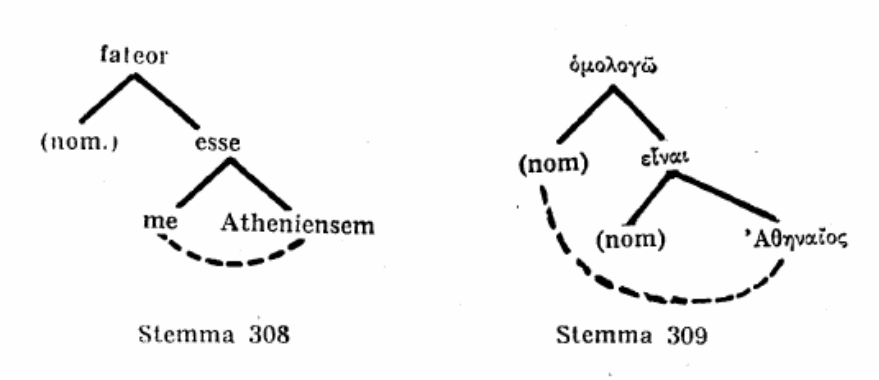


Figure 2.2: Stemmas demonstrating dropping of subject in Latin and Ancient Greek. In Ancient Greek, the verb "ὁμολογῶ" is in 1st person singular form. It is therefore clear from context that the subject is "I" and it is dropped from the sentence. Example is from Tesnière's Elements.

2.2.2.1 Head and dependent

The governing word is referred to as the **head** or governor of the relation and the subordinate word as the **dependent**. The relation is then represented as an arc directed from the head to the dependent. As an example, in Figure 2.3 "Meal" is the head of "Combo".

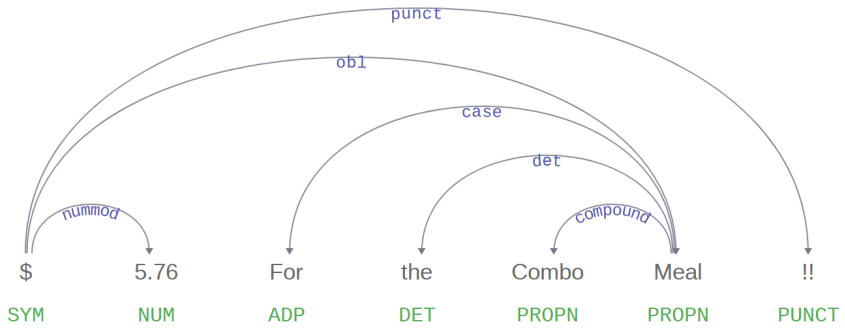


Figure 2.3: Example of a dependency graph in English.

2.2.2.2 Relations

The arcs are labelled with the type of relation that exists between the two words. We denote the set of all possible labels that can exist between any two words as L. As an example of a relation, in Figure 2.3, the relation between "Meal"

and "Combo" is "compound", denoting that "Combo" modifies "Meal" to create a compound nominal: "Combo Meal"[4]. A comprehensive listing of the possible labels and their meaning can be found in Appendix 1.

2.2.2.3 Dependency Graph

Definition 2. *A dependency graph G for a sentence $w_1, w_2 \dots w_n$, is a labelled directed graph with nodes V and arcs E with labels from the set L.*

$$G = (V; E) \quad \textit{where} \qquad V \subseteq \{w_0, w_1 \dots, w_n\}$$
$$E \subseteq V \times L \times V$$

where w_0 is a fictional *ROOT* node that dominates all the other nodes in the graph. In Figure 2.3 the fictional root node, if depicted, would be the head of "Meal". Furthermore, we give an example of what some elements of V and E would be for this example:

$$V \supset \{\text{ROOT}, \$, 5.76, \text{Meal}, !!\}$$
$$E \supset \{(\text{ROOT}, \text{root}, \text{Meal}), (\$, \text{nummod}, 5.76), (\$, \text{punct}, !!)\}$$

A dependency graph is also conventionally constrained to have the following two properties:

1. It must be acyclic.

2. Each node must only have a single head.

We note that the second constraint is important for us, because our graph parser we will discuss in Section 3 relies on it to make a single prediction for each word in the sentence.

2.2.2.4 Yield of a node

Definition 3. *The yield of a node w_i is the set of nodes dominated by w_i.*

The yield of a node w_i also contains itself.

[4]This was purely accidental.

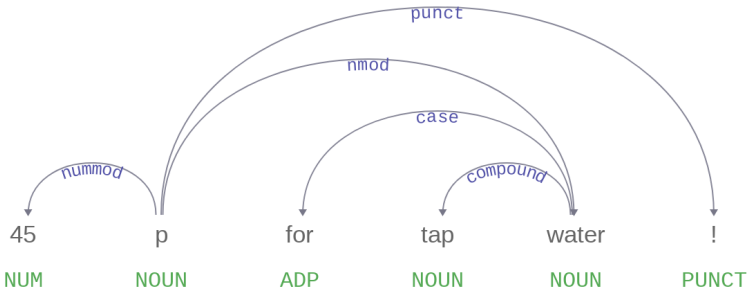
2.2.2.5 Projectivity

Projectivity is an additional constraint on the form of a dependency graph that limits which arcs can be drawn between nodes given their position in the sentence Kuhlmann and Nivre (2006).

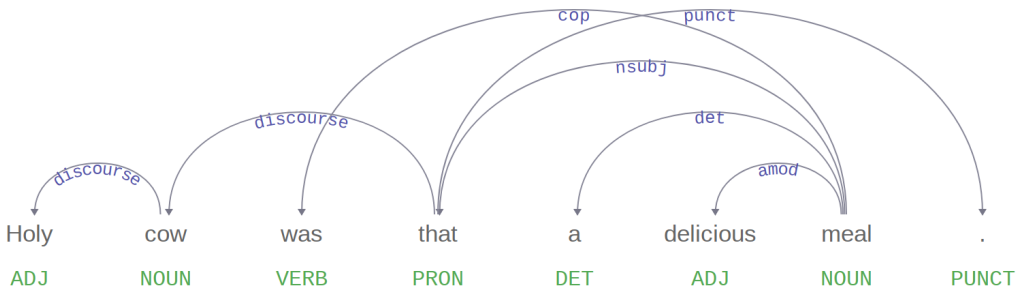
Definition 4. *A dependency graph is projective if the yield of all of its nodes are intervals.*

whereby an interval with endpoints j and k:

$$[j,\, k] := \{i \mid w_i \in V \;\wedge\; j \leq i \leq k\}$$



(a) Projective



(b) Non Projective—note the clear crossing arcs due to the punct arc.

Figure 2.4: Example of a projective dependency graph and a non-projective dependency graph from the Universal Dependencies dataset.

As an example, the dependency graph in Figure 2.4a is projective, while that in Figure 2.4b is non-projective. To see why, we note that the yield of "that" is $\{Holy, cow, that, .\}$, which have indices $\{1, 2, 4, 8\}$. This set is not an interval, since it contains gaps; (3, 5, 6 and 7) are missing.

A visual way to check for non-projectivity, is to check for arc crossings in the dependency graph. However, we need to be careful that the following two conditions are met:

- The nodes are ordered according to the words in the sentence (usually true)
- The artificial root node is illustrated and joined with the root of the dependency graph with an arc.

The projectivity constraint is usually satisfied for languages whose syntax enforces a relatively rigid word order, English being such an example. On the other hand, languages that do not impose constraints on the order of words in the sentence, such as Latin and Ancient Greek, have a high degree of non-projectivity.

2.2.3 Graph based dependency parsing

In general, approaches to dependency parsing can be coarsely classified into transition based approaches and graph based approaches. In this dissertation our focus is on graph based approaches, however we shall give a brief comparison of the two here.

Transition based parsers (Yamada and Matsumoto, 2003; Nivre and Hall, 2005; Chen and Manning, 2014; Alberti et al., 2017) treat the parsing problem as a search over a set of possible parsing actions. They enforce acyclic and projective constraints through the transition system they employ. A compelling characteristic they have is their efficiency; most projective transition based parsers are linear in the size of the input regarding runtime and memory.

Graph based parsers McDonald et al. (2005); Martins et al. (2010); Zhang et al. (2016); Dozat and Manning (2016) on the other hand, approach the parsing problem as a choice of which pairs of nodes to draw arcs between. A drawback of graph parsers compared to transition based parsers is that they typically have n^2 complexity[5]. This makes them computationally unattractive, especially given that long sentences unfortunately are not uncommon. However, graph based parsers can handle highly non-projective languages more effectively than transition based ones.

As opposed to transition based parsers, graph based parsers enforce tree and projectivity constraints by post-processing. The $n \times (n + 1)$ matrix of weights corresponding to all possible arcs in a sentence of n words is processed with a maximum spanning tree algorithm. When we want to constrain the output of a dependency graph to be projective, we use Eisner's algorithm (Eisner, 2000) which

[5]assuming they are 1st order graph parsers

runs in n^3 time. If we want the output to be acyclic but want to allow for non-projectivity, we employ the Chu Liu Edmonds algorithm (Chu, 1965; Edmonds, 1967), which is an n^2 algorithm.

The simplest graph based parsing approach—and the approach our parser follows—is arc factored; each arc is scored independently from all others. More elaborate but at the same time more computationally expensive approaches exist that take into account more context (Martins et al., 2013).

2.2.4 Neural network approaches for dependency parsing

While there had been earlier parsers employing neural networks, Chen and Manning (2014) was an influential paper that made neural based parsers fashionable. They proposed a neural transition based parser that leveraged word embeddings, part of speech embeddings and arc label embeddings. The compelling characteristics of this parser were its speed and that no manual feature engineering was needed to obtain good results. The authors argued that for non-neural approaches, speed was hampered by computing the large complex set of feature representations. A peculiar suggestion in that paper was the use of an x^3 activation function which didn't seem to catch on.

The next generation of neural transition parsers started employing recurrent Long Short Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) to make the embeddings context dependent (Dyer et al., 2015). At the same time, neural graph based parsers employed Bidirectional Long Short Term Memory Units (BILSTM) (Graves and Schmidhuber, 2005) to encode sentences (Kiperwasser and Goldberg, 2016; Zhang et al., 2016; Cheng et al., 2016).

Most recent work has built on the aforementioned models by employing subword representations (Kim et al., 2015; Ling et al., 2015). Our graph based parser we shall discuss in Section 3 also goes down this route. We follow Vania and Lopez (2017) who employed the aforementioned subword representations as well as simpler composition functions, such as addition, for language modelling. They report that character trigrams composed with a LSTM were very effective. However, they found that these representations are not substitutes for a morphological analysis, as when they used such information they significantly outperformed the subword models.

On a similar note, Yu and Vu (2017) compare such subword representations,

but for a transition based parser. They report that character convolutional networks outperform character BILSTMs, especially for agglutinative languages. Similar to our findings, they demonstrate that the subword model's improvement over the word model is in large part due to its ability to alleviate out of vocabulary issues.

Approaches to parsing in the past have mostly relied on using part of speech (Zhang et al., 2016; Kiperwasser and Goldberg, 2016; Bohnet and Nivre, 2012; Cheng et al., 2016) and morphological information (Lee et al., 2011; Bohnet et al., 2013; Cheng et al., 2016) as input. With the shift to neural network models, the community has found that jointly learning several tasks is beneficial. Many recent approaches jointly learn to predict part of speech tags (Chorowski et al., 2016; Ma and Hovy, 2017; Nguyen et al., 2017; Alberti et al., 2017).

There have been some interesting approaches for parsing that leverage language models. Vinyals et al. (2015) model constituency parsing as machine translation of brace delimited strings using an encoder/decoder model. Choe and Charniak (2016) use a language model to rerank parses from a constituency parser obtaining state of the art results for English.

In the spirit of Collobert et al. (2011), Hashimoto et al. (2016) train a neural network on very diverse tasks and obtain reasonable results on many, including dependency parsing.

2.3 Datasets & Evaluation

2.3.1 Datasets

Most recent work (Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Zhang et al., 2016; Dozat and Manning, 2016) report results on the Penn Treebank (PTB) and Chinese Treebank (CTB) datasets. However, English and Chinese aren't morphologically-rich. We therefore chose to evaluate our models on the CONLL 2006 (Buchholz and Marsi, 2006) and Universal Dependencies v1.3 (Nivre et al., 2016) datasets. The first was chosen so that we could compare to Zhang et al. whose architecture we followed to a great extent. The second was chosen since it is more recent and provides a larger selection of languages. Related work to which we shall be comparing to in Section 4 is listed in Table 2.1.

CONLL-2006	Universal Dependencies v1.3
Ballesteros et al. (2015)	Alberti et al. (2017)
Zhang et al. (2016)	Chorowski et al. (2016)
Cheng et al. (2016)	

Table 2.1: Work we will compare to on the CONLL 2006 and Universal Dependencies v1.3 datasets

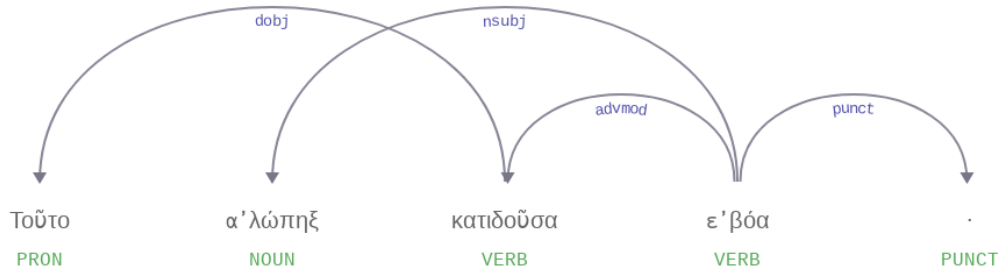
2.3.1.1 CONLL 2006

The CONLL 2006 dataset was introduced in the CONLL 2006 task (Buchholz and Marsi, 2006). For the challenge, submissions were run on 13 languages; we will be evaluation on Czech, German and Turkish. The datasets for these languages were converted automatically from constituency treebanks [6], phrase treebanks or other dependency formats to a common dependency treebank form. The form of the common dataset followed Joakim Nivre's Malt-TAB format Nivre and Hall (2005), a predecessor of the format that can be seen in 2.5. An important note here is that unlike the Universal Dependencies datasets introduced later on, the CONLL 2006 dataset has different tags for each language.

Overall a big difference in average labelled attachment score for various languages was reported among all submissions, suggesting that some morphologically-rich languages were harder to parse than others. As an example the average labelled attachment score for Turkish over all submissions was 56.0% while for Japanese it was 85.9% The two best submissions were a transition based parser by Nivre and a Graph parser by Mcdonald with average labelled attachment score on the 13 languages being 80.3 and 80.2 respectively McDonald and Nivre (2007). Both of these parsers turned out to be very influential for future dependency parsers.

The CONLL 2006 dataset has no standard train and development split. We therefore split the training set into 95% train and 5% development, as carried out by Cheng et al. (2016). This leads to a larger development set than that used in Zhang et al., where the last 374/367 sentences were used as development sets for Czech and German respectively.

[6] Rules were used to find the head of each constituent



(a) Dependency parse we revisit from the introduction.

```
1 Τοῦτο οὗτος PRON p–s——na– Case=Acc|Gender=Neut|Number=Sing 3 dobj _ _
2 ἀλώπηξ ἀλώπηξ NOUN n–s——fn– Case=Nom|Gender=Fem|Number=Sing 4 nsubj _ _
3 κατιδοῦσα κατεῖδον VERB v–sapafn– Case=Nom|Gender=Fem|Tense=Aor 4 advmod _ _
4 ἐβόα βοάω VERB v3siia—— Aspect=Imp|Number=Sing|Person=3|Tense=Past 0 root _ _
5 · · PUNCT u———— _ 4 punct _ _
```

(b) CONLL file format for the above dependency parse.

Figure 2.5: Elaboration on CONLL file format. The dependency parse we revisit from the introduction is represented with the above file in CONLL format. The columns are in the following order: id, form, lemma, upostag, xpostag, feats, head, deprel, deps, misc. We removed some of the morphological features (feats) from the above in order to fit the file onto the page.

2.3.1.2 Universal Dependencies

The Universal Dependencies (Nivre et al., 2016) dataset was compiled to encourage the development of dependency parsing across many languages. It was constructed with experience from previous challenges (Buchholz and Marsi, 2006; Nilsson et al., 2007; Tsarfaty et al., 2010) using a common universal tagset for all languages.

In the CONLL 2017 dependency parsing challenge (Zeman et al., 2017) the main theme was to develop dependency parsers on the freshly released Universal Dependencies v2.0 dataset in a real world setting. This meant that the input to the parsers at test time were tokens split by UDPipe (Straka and Straková, 2017) as opposed to the gold tokens. Additionally, participants were provided with the raw text of the sentences and given the choice of using their own tokeniser if they wanted to. More importantly, the part of speech tags, lemmas and morphological

annotations provided at test time were the output of UDPipe and not the gold ones. This meant that there systems that could provide output than UDPipe had a clear advantage.

A second axis the challenge expanded on was that of low resource languages. Some languages had scarce data and most participants attempted to build multilingual models in the spirit of Ammar et al. (2016) to improve the performance on such datasets. Additionally, the challenge included "surprise languages" which were not revealed until a week before the deadline for the submission.

The best overall system was that of Dozat et al. (2017), a system based on their model we described earlier (Dozat and Manning, 2016). Their biaffine attention graph parser was modified to additionally use character level word embeddings similar to Ballesteros et al. (2015) concatenated with their word embeddings and part of speech tag embeddings. They used their own part of speech tagger— a separate model—and argued that the fact that their part of speech tagger was more accurate than the UDPipe can affect results.

2.3.2 Evaluation

In order to evaluate dependency parsing methods, it has become the norm to report two metrics. Unlabelled attachment score (UAS) and Labelled attachment score (LAS).

$$UAS = \frac{\#correctarcs}{\#arcs} \tag{2.11}$$

$$LAS = \frac{\#(correctarcs \cap correctlabel)}{\#arcs} \tag{2.12}$$

UAS measures the percentage of arcs of the dependency graph a parser gets correct when not taking the labels into account. LAS is the percentage of arcs a system gets correct when also taking the labels into account. It is important to note that for the CONLL 2006 dataset punctuation is not accounted for when computing LAS and UAS.

In an unpublished paper Nivre (2016) has expressed concern that these metrics for evaluation aren't fair when comparing between multiple languages. This is because languages with a large number of function words get higher scores because function words are usually much easier to label. While this may be a valid point, using modified versions of the metrics would make our comparison to earlier work harder.

3

Implementation

In this section we begin by introducing the graph dependency parser we implemented based on (Zhang et al., 2016) and Kiperwasser and Goldberg (2016), that leverage Bidirectional Long Short Term Memory networks (BILSTM) (Graves and Schmidhuber, 2005) to encode the words of each sentence. We then describe our choices for preprocessing and motivate the choice of hyperparameters used for training each of the three proposed variants of the model.

We note that by reimplementing all models we will be exploring using the same framework (Tokui et al., 2015), the comparison is made on as equal a footing as possible; there are no effects of choice of framework and implementation differences that may otherwise bias results towards a specific model.

From here onwards we assume familiarity with machine learning and neural network techniques. Due to space and time constraints, we haven't included an introductory section covering such topics. For a comprehensive introduction to neural networks for natural language processing applications, we refer the reader to Goldberg (2016).

We try to follow notation in the Nomenclature summarised before the introduction. Unless explicitly noted, lowercase characters (c) denote scalars , lowercase

bold characters ($\boldsymbol{v}$) vectors and capital boldface characters ($\boldsymbol{M}$) matrices. Vectors are column vectors and functions of vectors are assumed to retain this characteristic.

3.1 Graph based dependency parser

For ease of exposition, we break down our dependency parser conceptually into two components, the encoder and the parser.

The encoder's reason of existence is to give us a vector representation for each word in the sentence. As we shall see in the next sections, these representations can be constructed in several ways. Hence, we shall present three models that differ in the way they construct representations for the words.

Returning to the second component, as hinted the parser does not differ based on the choice of encoder. It uses the aforementioned word representations constructed from one of the proposed encoders to make predictions and form the dependency graph of the sentence.

We begin by introducing the common component of all our models, the parser, deferring the description of the encoder to the next section. Next, we describe how the sentence encoder works with word embeddings, utilising no subword information. We shall refer to the parser that uses this encoder as word-BILSTM. Finally, we introduce two subword encoders that provide alternative ways of constructing the word embeddings compositionally from characters. They employ two different neural architectures as composition functions, a BILSTM and a convolutional neural network (CNN) (LeCun et al., 1990) respectively. These alternative representations can be used as drop in replacements for the character unaware word embeddings in the sentence encoder, leading to what we shall henceforth refer to as the char-BILSTM encoder and the char-CNN encoder.

3.1.1 Parser

As we have discussed in Section 2.2, a dependency parser accepts as input a sentence and outputs a labelled dependency graph which embodies the syntactic relationship between the words in the sentence. Our parser therefore needs to complete two tasks. First it needs to decide which pairs of words are in a head/dependent relationship—which arcs to draw[1]. Second, for each such arc it needs to predict

[1]We shall refer to this as arc prediction and head prediction interchangeably.

a label denoting the relationship that exists between the head and the dependent word.

For arc prediction, we follow (Zhang et al., 2016) and model dependency parsing as head selection. Namely, for each word in the sentence we predict which word should be its head. This approach is viable since, as we discussed in Section 2.2.2.3, each word in a dependency graph has a single head. The head could be any of the other words in the sentence or an artificial root symbol. In the latter case, the word is interpreted as being the head of the sentence. An important point here is that this unconstrained approach leads to output that may contain cycles. However, it is clear from the literature that the BILSTM model learns to actually predict trees in the vast majority of cases even having been trained without such a constaint (Zhang et al., 2016; Chorowski et al., 2016) (see Section 2.2.3 for details and Table 4.4 for more evidence on this point).

For label prediction, we use information about the head and the dependent joined by the arc to make a decision about the label. At training time we take into account the annotated arcs in the training set. At test time we count on the arcs the system predicted.

To accomplish both head and label prediction, we form a probability distribution over possible heads in the first case and the vocabulary of labels in the other. Our model following Kiperwasser and Goldberg (2016) obtains these probability distributions using two neural networks with a hidden layer, one network for each task. These are trained jointly with one of the encoders we shall discuss in the following sections.

For the arc prediction case, using this neural network we compute an activation for each possible pair. Note that as hinted earlier, the head word can be the *ROOT* representation while the dependent cannot.

$$\begin{gathered} a(\boldsymbol{e}_i, \boldsymbol{e}_j) = \boldsymbol{v}_a^T \, tanh(\boldsymbol{D}_a \boldsymbol{e}_i + \boldsymbol{H}_a \boldsymbol{e}_j + \boldsymbol{b}_{a_1}) + b_{a_2} \\ \underbrace{i \in \{1, 2 \ldots N\}}_{\text{dependent}} \quad \underbrace{j \in \{0, 1 \ldots N\}}_{\text{head}} \end{gathered} \tag{3.1}$$

where $\boldsymbol{H}_a, \boldsymbol{D}_a \in \mathbb{R}^{|\boldsymbol{e}| \times |\boldsymbol{e}|}$, $\boldsymbol{v}_a, \boldsymbol{b}_{a1} \in \mathbb{R}^{|\boldsymbol{e}|}$ and b_{a2} is a scalar. Given these $n \times (n+1)$ activations we compute the probability that w_j is the head of w_i as follows.

$$P_{head}(w_j | w_i, S) = \frac{exp(a(\boldsymbol{e}_i, \boldsymbol{e}_j))}{\sum_{k=0}^{N} exp(a(\boldsymbol{e}_i, \boldsymbol{e}_k))} \quad i \in \{1, 2 \ldots n\} \tag{3.2}$$

For label prediction the network is similar, but we only predict labels for head-dependent pairs that are joined by an arc. Hence we restrict the contextual

	Zhang et al. (2016)	Kiperwasser and Goldberg (2016)	word-BILSTM
Loss	crossentropy	hinge	crossentropy
Tree constraint	test	train and test	test
Label prediction	separate network	joint	joint
BILSTM units	300	125	200
Embeddings	300	100	200
Dropout	Dropout	Word Dropout	Dropout
Training tricks	–	Loss augmented inference	–

Table 3.1: Comparison of Zhang et al. (2016) to Kiperwasser and Goldberg (2016) and our hybrid implementation.

embeddings of interest to heads and dependents, $\boldsymbol{e}_h$ and $\boldsymbol{e}_d$ respectively.

$$l(\boldsymbol{e}_h, \boldsymbol{e}_d) = \boldsymbol{W}_l \, tanh(\boldsymbol{D}_l \boldsymbol{e}_d + \boldsymbol{H}_l \boldsymbol{e}_h + \boldsymbol{b}_{l_1}) + \boldsymbol{b}_{l_2} \tag{3.3}$$

where $\boldsymbol{H}_l, \boldsymbol{D}_l \in \mathbb{R}^{|\boldsymbol{e}| \times |\boldsymbol{e}|}, \boldsymbol{W}_l \in \mathbb{R}^{|V_l| \times |\boldsymbol{e}|}, \boldsymbol{b}_{l_1} \in \mathbb{R}^{|\boldsymbol{e}|}, \boldsymbol{b}_{l_1} \in \mathbb{R}^{|V_l|}$ and $|V_l|$ is the size of the label vocabulary. We then predict the label using a softmax.

$$P_{label}(l_i | w_h, w_d, S) = \frac{exp(l(\boldsymbol{e}_h, \boldsymbol{e}_d)[i])}{\sum_{i=1}^{|V_l|} exp(l(\boldsymbol{e}_h, \boldsymbol{e}_d)[i])} \tag{3.4}$$

The network is trained to minimise the joint loss from label prediction and head prediction.

$$J_{head}(\theta) = -\frac{1}{|T|} \sum_{S \in T} \sum_{i=1}^{N_S} \log P_{head}(gold(w_j) | w_i, S) \tag{3.5}$$

$$J_{label}(\theta) = -\frac{1}{|T|} \sum_{S \in T} \sum_{i=1}^{N_S} \log P_{label}(gold(l_i) | gold(w_j), w_i, S) \tag{3.6}$$

$$J_{total}(\theta) = J_{head}(\theta) + J_{label}(\theta) \tag{3.7}$$

Since we have used ideas from both Zhang et al. (2016) and (Kiperwasser and Goldberg, 2016) in Table 3.1 we summarize the differences between the two models and our hybrid.

3.1.2 Word encoder

Our approach for the word level encoder follows Zhang et al. (2016) and Kiperwasser and Goldberg (2016), where the word and part of speech tags of the sentence are first mapped to embeddings and then fed into a BILSTM neural network to obtain context dependent embeddings.

More concretely, given an input sentence of words $w_1, w_2, \ldots, w_n$, we first prepend a root symbol w_0 to it. We do so since we need to create a representation for the root if we are to be able to predict that a given word is the head of the sentence (it has this fictional root symbol as its head). We also pad the sentence on the left with the start sentence symbol <s> and on the right with the end of sentence symbol </s> [2] such that the position of the word in the sentence is explicit with respect to the sentence boundaries. This is done to add more information about word position to the benefit of the BILSTM network encoder which we shall describe shortly.

In order to obtain representations for the words, we first create a word vocabulary with a prespecified capacity $|V_w|$. The vocabulary is then built by retaining the $|V_w|$ most common words in the training set. Ties for words that occurred a number of times equal to what turned out to be the threshold are broken depending on which of these words gets processed first. In a similar fashion, we create vocabularies for part of speech tags and arc labels. The difference is that each distinct symbol in the training set is assigned an id and there is no prespecified capacity or dependency on occurrence frequencies.

The constructed vocabularies all assign an id to each token. For each word in the sentence we replace words that didn't make it into the vocabulary with a reserved token for unknowns: *UNK*. This is treated as any other token hereafter. More concretely, in Table 3.2 we list the ids we have preallocated for our vocabularies. Some of these tokens such as *PAD* and <w>, </w> are only used in the subword encoders we shall describe in the next sections.

Next, we create an embedding for each entry in the corresponding vocabularies. As an example, for words we create an embedding matrix $\boldsymbol{E}_w \in \mathbb{R}^{d_w \times |V_w|}$, where $|V_w|$ is the size of the word vocabulary V_w and d_w the number of learnable weights we assign to each word. For each sentence we wish to encode, each word w is

[2] Arguably the start sentence symbol may not be needed since the root symbol is always prepended to the beginning.

Reserve	ID	Description	Token	ID	Description	Token
	0	start sentence	<s>	3	start word	<w>
Always	1	end sentence	</s>	4	end word	</w>
	2	root symbol	w_0	5	pad	*PAD*
Word/Char Vocab				6	unknown	*UNK*

Table 3.2: Vocabulary ids we preallocate for specific tokens. We shall refer to these within the text using the representation in the Token column. As an example, the most common word in the vocabulary will get the id 7, while for part of speech tags start from id 6. This is so because we assume all possible tags must exist in the training set—namely that there is no reason to allow for unknown tags.

looked up in the embedding matrix $\boldsymbol{E}_w$, an operation we shall denote by $e_w(w)$. If we are also using other information such as part of speech tags t we similarly map them to embeddings $e_t(t)$ from the embedding matrix $\boldsymbol{E}_t$. The input to the network for each word is a concatenation of these vectors. For our default word level model, which only employs words and their part of speech, we embed each word of the augmented sentence w_i and its part of speech tag t_i as follows.

$$\boldsymbol{e_i} = \left[e_w(w_i); e_t(t_i)\right] \tag{3.8}$$

Assuming the dimensionality of the embedding vectors for words and part of speech is $|e_w(w)|$ and $|e_p(p)|$ respectively, the concatenated embedding $\boldsymbol{e}$ would have dimensionality $|e_w(w)|+|e_p(p)|$. According to the above, we encode an input sentence <s>, $w_0, w_1, \ldots, w_n$, </s> with part of speech tags <s>, $t_0, t_1 \ldots t_n$, </s> to get $n+3$ vectors of dimensionality d_w+d_p, the embeddings $\boldsymbol{e}_{\text{<s>}}, \boldsymbol{e}_0, \boldsymbol{e}_1, \ldots \boldsymbol{e}_n, \boldsymbol{e}_{\text{</s>}}$.

We could use these embeddings as input to our parser, however they have no information relating to the position of the word in the sentence. As any sane person who hasn't worked with bag of word models for too long knows, word order matters in human language. In order to capture temporal information and condition the word representations on previous words in the sentence, we feed the embeddings into a BILSTM.

$$\boldsymbol{h}_i^F, \boldsymbol{c}_i^F = \overrightarrow{LSTM}^F(\boldsymbol{e}_i, \boldsymbol{h}_{i-1}^F, \boldsymbol{c}_{i-1}^F) \tag{3.9}$$

$$\boldsymbol{h}_i^B, \boldsymbol{c}_i^B = \overleftarrow{LSTM}^B(\boldsymbol{e}_i, \boldsymbol{h}_{i+1}^B, \boldsymbol{c}_{i+1}^B) \tag{3.10}$$

where $\boldsymbol{h_i}$ is the output and $\boldsymbol{c}_i$ is the internal cell state of the LSTM when processing the i^{th} word. The forward LSTM ($\overrightarrow{LSTM}^F$) encodes the words of the sentence from left to right and the backward LSTM ($\overleftarrow{LSTM}^B$) from right to left. Note that the forward and backward LSTMs don't share any weight matrices, they are independent neural networks. The word representation for w_i conditioned on the sentence is then obtained by concatenating the forward and backward states $\boldsymbol{h}_i^F$ and $\boldsymbol{h}_i^B$ output when encoding w_i. At this point we dispose of the activations of the start (<s>) and end (</s>) sentence tokens, since their sole purpose was to mark the boundaries of the input for the BILSTM, hence $i \in [0, n]$ when considering the concatenation of the output states.

$$\boldsymbol{a}_i = \left[\boldsymbol{h}_i^F; \boldsymbol{h}_i^B\right] \qquad i \in [0, n] \tag{3.11}$$

In this approach we concatenate the forward and backward LSTM activations for each input to get a vector of twice the length. It is important to note that other choices have been explored in the literature for combining the BILSTM states, such as element-wise addition (Chorowski et al., 2016) and transformation with an additional neural network layer (Ling et al., 2015).

We have hitherto shown how the encoder constructs a vector representation for each sentence $w_0, w_1, \ldots, w_n$ as $n+1$ vectors $\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$. This representation is used as input to the parser component we shall elaborate on in Section 3.1.1.

The aforementioned sentence encoding doesn't take into account subword information. In the following section we present two ways of constructing word embeddings compositionally from characters. These can be used as drop in replacements for the word embeddings $e_w(w)$ of Equation 3.8 to obtain a sentence representation that is character aware.

3.1.3 Subword encoders

In this section we shall explore two subword encoders. The first composes character embeddings using a BILSTM. The second employs a CNN, learning feature detectors (filters) over windows of character n-gram embeddings.

We shall first discuss how we process the words in both cases since this step is shared. Then we shall describe how each subword encoder constructs the word embeddings. Note that after we build the word embeddings, we make them context sensitive following exactly the same steps as the word level encoder. Hence,

the representations we elaborate on in the next two sections are used as substitutes for $e_w(w_i)$ in Equation 3.8.

The character vocabulary is constructed in exactly the same manner as the word vocabulary. Given a prespecified character vocabulary size $|V_c|$, we retain the $|V_c|$ most common characters of the training set and map all other characters to *UNK*.

With the same justification as for the word case, we pad the characters of each word with the start word <w> and end word </w> ids which we have reserved in our character vocabulary (see Table 3.2 for details).

We then construct a character embedding matrix $\boldsymbol{E}_c \in \mathbb{R}^{d_c \times |V_c|}$, where $|V_c|$ is the size of the character vocabulary and d_c is the number of learnable weights we want each character to have.

As noted in Ballesteros et al. (2015), we can take advantage of the fact that many words are common and only compute their embedding once and cache the result. This can lead to a significan speed up, which is useful since these subword models are typically much slower to train than the word level models. One caveat is of course that we can only do this for each batch at training time, since the embeddings get updated after error backpropagation. We therefore clear the cache after each batch. We cache word embeddings both for the BILSTM and the CNN subword encoders. We found that after caching, the char-CNN model is approximately 1.5 times slower than the word-BILSTM model. The char-BILSTM model is approximately 2.5 times slower than the word-BILSTM model.

We now proceed to discuss how the subword models construct the word representations.

3.1.3.1 char-BILSTM subword level encoder

The char-BILSTM encoder we describe here was proposed in Ling et al. (2015) and has been employed for dependency parsing in Ballesteros et al. (2015). The BILSTM is employed in the same way as the sentence encoder to obtain context dependent character embeddings. However, in this case we want a single output representation for the whole word, which would correspond to us wanting a representation for the sentence instead of the words in the sentence embedder. For this reason we only take the output of the BILSTM after processing the last character of the word, as we shall elaborate shortly.

Given the character representation of a word <w>, $c_1 \dots c_n$, </w> we look up each character in the embedding matrix $\boldsymbol{E}_c$ to get the embeddings

$\boldsymbol{e}_{\text{<w>}}, \boldsymbol{e_1} \ldots \boldsymbol{e_n}, \boldsymbol{e}_{\text{</w>}}$. We then encode each character embedding using a BILSTM.

$$\boldsymbol{h}_i^F, \boldsymbol{c}_i^F = \overrightarrow{LSTM}^F(\boldsymbol{e}_i, \boldsymbol{h}_{i-1}^F, \boldsymbol{c}_{i-1}^F) \tag{3.12}$$

$$\boldsymbol{h}_i^B, \boldsymbol{c}_i^B = \overleftarrow{LSTM}^B(\boldsymbol{e}_i, \boldsymbol{h}_{i+1}^B, \boldsymbol{c}_{i+1}^B) \tag{3.13}$$

However, we only take the output of the BILSTM at the last step. That is, the output of the forward LSTM($\overrightarrow{LSTM^F}$) after processing the end word character $\boldsymbol{h}_{\text{</w>}}^F$ and the output of the backward LSTM($\overleftarrow{LSTM^B}$) after processing the start word character $\boldsymbol{h}_{\text{<w>}}^B$. The word representation is then computed by concatenating these two outputs [3].

$$\boldsymbol{a}_w = \left[\boldsymbol{h}_{\text{</w>}}^F; \boldsymbol{h}_{\text{<w>}}^B\right] \tag{3.14}$$

As we have alluded to earlier, this word embedding $\boldsymbol{a}_w$ is then used as input to the sentence encoder described in the previous section to obtain a context sensitive embedding before being fed into the parser.

3.1.3.2 char-CNN subword level encoder

In this section we elaborate on the char-CNN encoder implementation proposed in Kim et al. (2015). In this approach a convolutional network is used to identify salient character n-grams and encorporate their presence into a word representation.

We first pad shorter words to the longest word found in the batch such that we can train using mini-batches. Yu and Vu (2017) and Ma and Hovy (2017) pad shorter words symmetrically on both sides. Kim et al. only pad to the right as we verified after checking the released code. We follow Kim et al. and pad shorter words to the right using the reserved *PAD* token, as this approach produces less windows that have padding overlapping with actual word input.

In our approach, instead of limiting the maximum size of a word, we split each batch of words into two batches, the first one containining the longest 0.1% of words. We note that in retrospect this wasn't a very good idea because the batch size affects the way padding is carried out. Nevertheless, we found the effect to be very small and didn't retrain all our models with one of the padding schemes mentioned earlier.

Once we have each of the words padded, we follow a similar procedure to that we carried out to embed the characters in the previous section for the BILSTM. For

[3] Ling et al. (2015) actually compute $tanh(\boldsymbol{W}\boldsymbol{h}_n^F + \boldsymbol{U}\boldsymbol{h}_n^B + \boldsymbol{b})$. for part of speech tagging, but in the parsing paper (Ballesteros et al., 2015) simply concatenate: $\left[\boldsymbol{h}_n^F; \boldsymbol{h}_n^B\right]$

each word given its character representation <w>, $c_1 \dots c_n$, </w>, PAD . . . we look up each character in the embedding matrix $\boldsymbol{E}_c \in \mathbb{R}^{d_c \times |V_c|}$ to get the embeddings for the word $\boldsymbol{e}_{\text{<w>}}, \boldsymbol{e_1} \dots \boldsymbol{e_n}, \boldsymbol{e}_{\text{</w>}}, (\boldsymbol{e}_{\text{PAD}})*$. We then stack together these embeddings as columns of matrix $\boldsymbol{C_w}$ such that we can slide our convolutional filters over them. To facilitate this operation we henceforth define that any indexes applied to the matrix act on the columns. We can do so since we will not need to deal with the dimensionality of the character embeddings—for our purposes here they are of fixed size and indivisible. At this point, to put the following steps into perspective, an illustration of the model we are reimplementing is depicted in Figure 3.1.

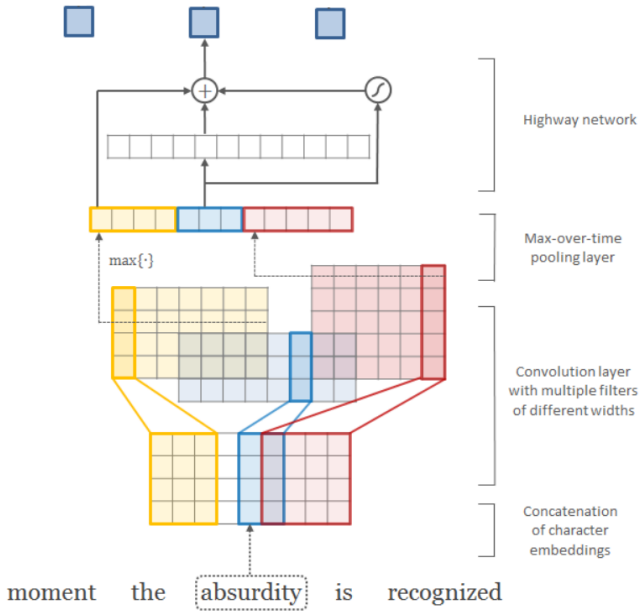


Figure 3.1: Illustration of char-CNN encoder architecture. The outputs drawn in blue are the word embeddings we will feed into our sentence encoder. The image is a modified version of the original we attribute to Kim et al. (2015).

In order to create a word representation, we slide a total of k filters $\boldsymbol{F}_j \in \mathbb{R}^{w_j \times d_c}, \quad j \in \{1, 2 \dots k\}$ each of a particular width w_j and fixed height equal to the character embedding dimension d_c. Each filter is slided with a stride of 1 over the stacked character embedding matrix $\boldsymbol{C_w}$. At each step we compute an activation for each filter $\boldsymbol{F}_j$. The activation at position i for filter $\boldsymbol{F}_j$ of the convolutional network is computed as so.

$$f_j[i] = tanh\big(\langle \boldsymbol{C_w}[i : i + w_j - 1], \boldsymbol{F}_j \rangle\big) \tag{3.15}$$

where $\boldsymbol{C_w}[i : i + w_j - 1]$ is a window of w_j columns of matrix C_w starting from column i and $\langle A, B \rangle = Tr(\boldsymbol{A}\boldsymbol{B}^T)$ is the Frobenius inner product[4].

We then perform max pooling over time. For each filter $\boldsymbol{F}_j$ we retain the maximum activation over all steps.

$$y_j = \max_i f_j[i] \tag{3.16}$$

Finally, in order to construct the embedding for our word we follow Kim et al. and pass the activation $\boldsymbol{y}$ through a highway network(Srivastava et al., 2015). The output of the highway network is the sum of two terms the contribution of which is controlled by a gate. The first term is an affine transform of the input followed by a non-linearity. In our case we used a tanh non-linearity.

$$g(\boldsymbol{y}) = tanh(\boldsymbol{W}_S\boldsymbol{y} + \boldsymbol{b}_S) \tag{3.17}$$

The second term is the input $\boldsymbol{y}$. The justification for this is that it allows the network to carry the input to the next layer thus allowing gradients to flow more freely during backpropagation without the hindrance of saturating non-linearities. The amount that each of these terms contributes is modulated by a gate. This gate is itself learned along with the network.

$$\boldsymbol{t} = \sigma(\boldsymbol{W}_T\boldsymbol{y} + \boldsymbol{b}_T) \tag{3.18}$$

where σ is the sigmoid function. The output of the network is thus:

$$\boldsymbol{a}_w = \underbrace{\boldsymbol{t} \odot tanh(\boldsymbol{W}_S\boldsymbol{y} + \boldsymbol{b}_S)}_{\text{transform}} + \underbrace{(1 - \boldsymbol{t}) \odot \boldsymbol{y}}_{\text{carry}} \tag{3.19}$$

This compositionally constructed word embedding $\boldsymbol{a}_w$ can then be used as input to Equation 3.8 as stated earlier.

3.2 Setup

In this section we elaborate on the nitty gritty implementation details of our neural network models which was carried out using the Chainer neural network framework(Tokui et al., 2015). We first describe how we preprocessed the input for the word-BILSTM and then for the subword models. Then we justify our choices of hyperparameters for these models based on experiments on the Universal Dependencies v1.3 development set. Lastly we elaborate on details regading the training procedure.

[4]Or if you speak numpy, np.dot(A.ravel(), B.ravel())

3.2.1 Preprocessing

For our models we first narrowed down our choices for preprocessing, since there are already ample hyperparameters to choose for each individual model. These choices therefore may not be optimal, since they were made when we had little experience training the models.

3.2.1.1 Word models

We carry out basic preprocessing to make the vocabulary smaller and combat spurious sparsity at the word level. We argue that this makes the comparison to models that can capture subword information more realistic, given that such preprocessing steps are generally used in word level approaches and are easy, alas tedious to carry out. The following transformations are carried out to each token in the following order.

1. Each token is lowercased.
   LOUDTEXT → *loudtext*
2. We replace standard numeric forms with the string `__NUM__`.
   `32$` → `__NUM__$`.
3. We truncate a string of 3 or more consecutive identical characters to 2.
   greeeeeeatt!!! → *greeatt!!* [5]

We also considered removing diacritics similar to our approach on the subword level which we shall describe in the following section. However, there are cases where this would produce the same representation for different words. As an example, in Modern Greek "κανείς" means "nobody", but "κάνεις" means "you do". We therefore did not remove diacritics to avoid such sources of ambiguity.

(3.20) *Κ*ανείς δουλειά σήμερα;
nobody work today?
"Is anybody working today?"

(3.21) *Κ*άνεις δουλειά σήμερα;
you do work today?
"Are you working today?"

3.2.1.2 Subword models

For our subword models we chose to lowercase all characters and expand diacritics. As can be seen in Figure 3.2, the effect of preprocessing for Ancient Greek isn't

[5] We are not aware of a language which uses 3 consecutive identical characters in words. The reason we added this function is because in the English section of Universal Dependencies v1.3 we have reviews and email: "it s prettty much the same $$$, please advice !?????!!!!"

large. We chose to lowercase and expand diacritics, because it performed equally well with the other preprocessing methods. At the same time it reduced the character vocabulary size. We note at this point, that we made this decision early on to narrow down our search space. Given our current models we would reconsider this step in future work, given that we only explored training for the first 20 epochs. As we will see in the next section, we later found hyperparameters with dropout for which the performance improved for many more epochs—well after 50.

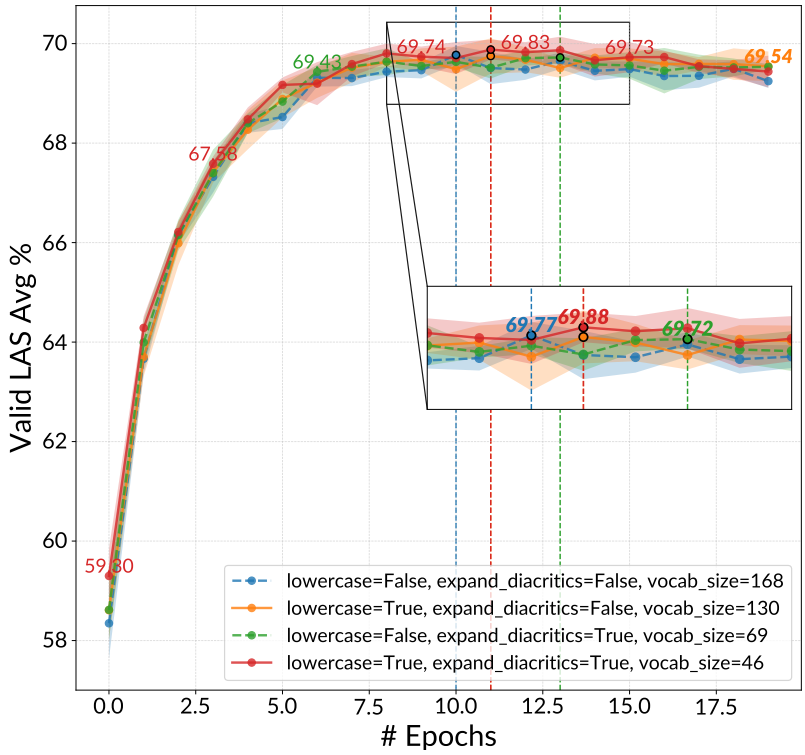


Figure 3.2: Effect of preprocessing on training for Ancient Greek during initial experiments. The plotted lines are the average of 3 runs with different seeds. The shading (blur) reports ±1 standard deviation from the mean.

3.2.2 Hyperparameter selection

In order to make as fair a comparison as possible between the models, we follow Collins et al. (2016) and use the same fixed budget of trainable parameters for each model. A similar approach was also followed by Kim et al. (2015), since they used the same number of total trainable parameters in their char-CNN language model and the word level language model they compared to.

In our case, we constrain the number of trainable parameters up until the

sentence encoder[6] to be the same, given that the model is shared from that point on. As an example, our char-BILSTM model can have $5000 \times 200 = 1000000$ trainable parameters in the character encoder network, given that we use a word vocabulary size of 5000 and each word embedding has 200 dimensions. In general, we note the following:

- The word-BILSTM model allocates a lot of trainable parameters on word embeddings due to the large vocabulary size.
- The char-BILSTM has very few trainable parameters in character embeddings due to the insignificant vocabulary size, and therefore can afford to have most of its parameters in the character BILSTM encoder.
- The char-CNN is very frugal with its parameters over the vocabulary, since convolutional neural networks don't have fully connected layers (large square matrices). Most of the parameters for this model are therefore distributed in the highway network[7].

Based on the above constraints we chose the number of layers, units and filters for our models as can be seen in Table 3.3.

Given the above restriction on our budget of trainable parameters we fine tuned the following hyperparameters in this order.

1. Select the batch size for each model.
2. Choose dropout for word-BILSTM.
3. Choose char-BILSTM and char-CNN dropout based on word-BILSTM values that worked best.

We tuned our batch size on three languages (English, Finnish and Ancient Greek) using three runs with different random seeds. We did so because the effect of random initialisation of the embeddings was not negligible, as reported in Weiss et al. (2015). We chose these languages as representatives of diverse language typologies. English is mixed, Finnish is agglutinative and Ancient Greek is fusional with free word order. For the following experiments we ran 50 epochs.

[6] We include the sentence encoder parameters, since if the output of a subword level encoder is of higher dimensionality, this affects the number of parameters in the sentence encoder BILSTM due to it having a rectangular input to hidden weight matrices.

[7] There may be more effective ways to distribute use these trainable parameters

	word-BILSTM	char-BILSTM	char-CNN
# parameters	2328000	2308000	2324100
vocabulary size	5000	≈ 100	≈ 100
embedding size	200	200	15
sent BILSTM layers	2	2	2
sent BILSTM units	200	200	200
char BILSTM layers	—	1	—
char BILSTM units	—	200	—
highway layers	—	—	1
filter ngrams	—	—	1,2,3,4,5,6
# filters per ngram	—	—	$20 + (n-1) \cdot 25^{\dagger}$
arc units $^{\diamond}$	100	100	100
label units $^{\diamond}$	100	100	100

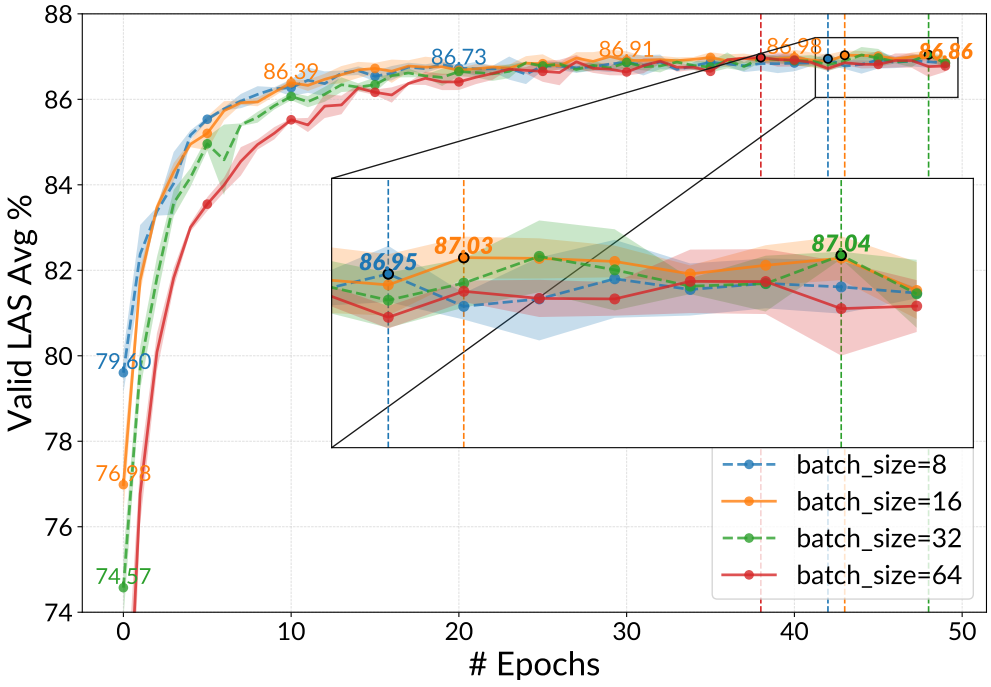
† Kim et al. (2015) use $n \cdot 25$ for their small model, namely $[25, 50, 75, 100, 125, 150]$.

$^{\diamond}$ These do not count towards the trainable parameter budget.

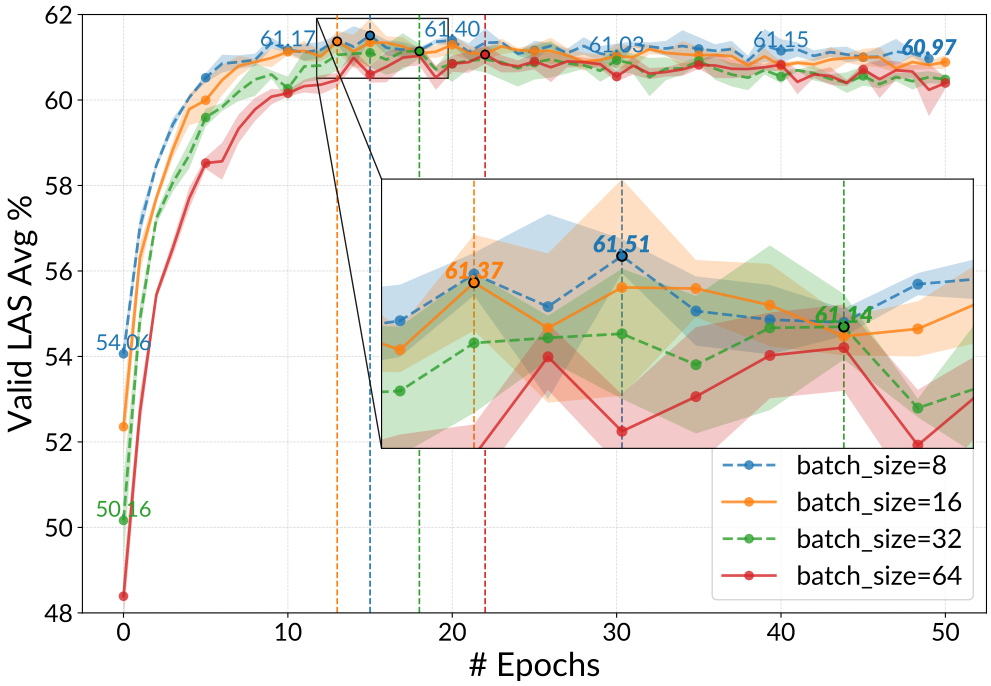
Table 3.3: Choice of layers, unit sizes and filters constrained by training parameter budget. For more details, the exact configuration used for training these models can be found in Figure 1 in the Appendix. We verified that the models had the same number of parameters by checking that the sizes of the serialized weight matrices were nearly the same (about 10 Mb).

3.2.3 Batch size

For the word-BILSTM we found that a batch size of 32 was a reasonable choice. As can be seen in Figure 3.3, the most glaring disparity is how much lower LAS the word model achieves for Ancient Greek compared to English (see Figure 2 in the Appendix for a comparison between English, Finnish and Ancient Greek side to side). It is apparent that morphologically-rich languages are harder to parse using a word level model. Moreover, it is clear from these initial results that optimising for a single language is not a good idea. On the other hand, taking into account more languages is not that straightforward either.



(a) Dev LAS for English



(b) Dev LAS for Ancient Greek

Figure 3.3: Batch size effect on word-BILSTM Labelled Attachment Score for English and Ancient Greek. The curves are the average of three runs with different random seeds. The shaded envelope of the curve is ±1 standard deviation.

The char-BILSTM showed similar trends to the word-BILSTM with an optimal batch size of 32. However, as expected the LAS curve was much higher and smoother on the development set. For the char-CNN we use a batch size of 16.

	word-BILSTM	char-BILSTM	char-CNN
batch size	32	32	16

Table 3.4: Choice of batch size for our models.

3.2.4 Dropout

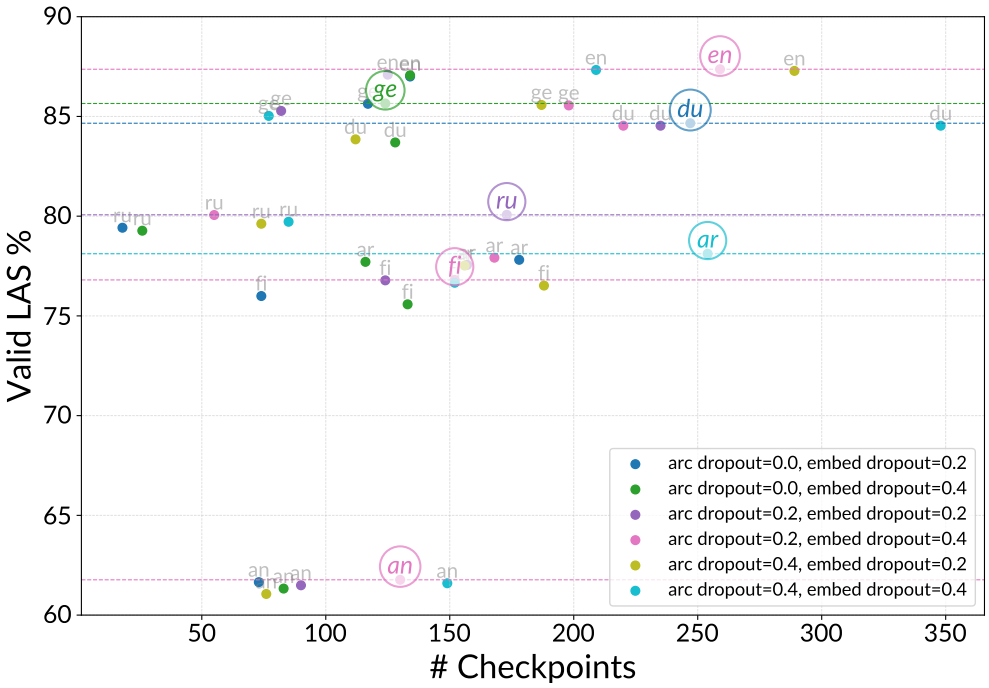


Figure 3.4: Effect of dropout on word-BILSTM Labelled Attachment Score. The best resulting experiment for each language is circled. Arc dropout is dropout applied to the input of the arc prediction network. Embed dropout is dropout applied to the concatenated word and part of speech embedding. Languages are from top to bottom: English, German, Dutch, Russian, Arabic, Finnish and Ancient Greek.

Instead of running experiments with different seeds, for choosing dropout we instead evaluated on many different languages to sense if there are any general trends. The combinations we tried were based on initial experiments and reported

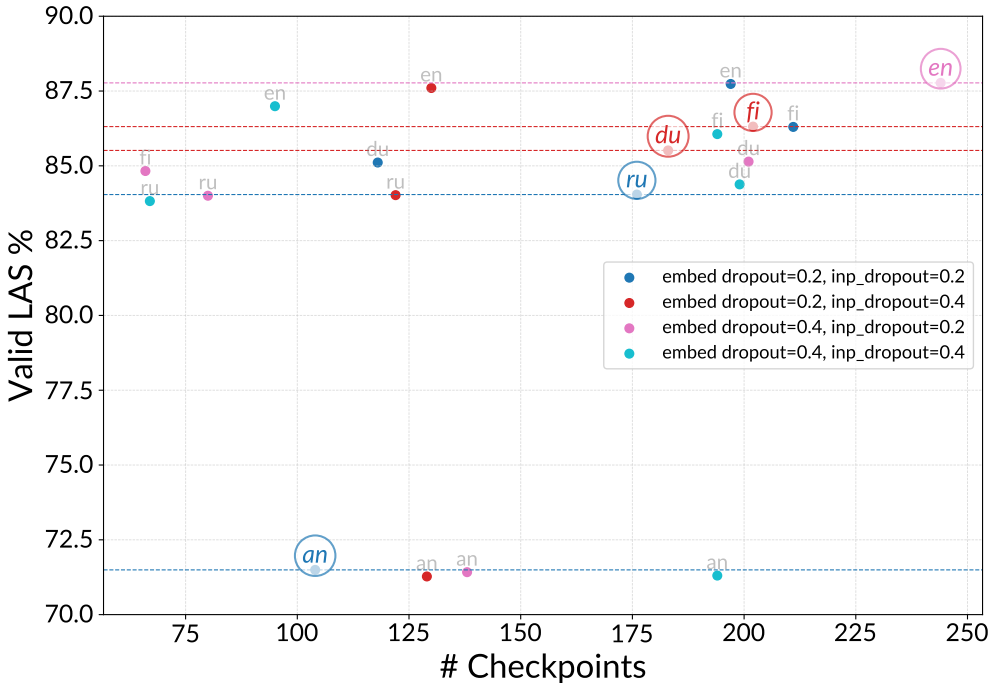


Figure 3.5: Effect of dropout on char-BILSTM Labelled Attachment Score. The best resulting experiment for each language is circled. Embed dropout is dropout applied to the concatenated word and part of speech embedding. Inp dropout is the dropout applied to the character embeddings before feeding them into the char-BILSTM. Languages are from top to bottom: English, Finnish, Dutch, Russian and Ancient Greek.

settings in the literature. Based on the results in Figures 3.4,3.5 and 3.6, we choose values for dropout as summarised in Table 3.5.

3.2.5 Training

3.2.5.1 Weight initialisation

Initialisation of all linear layers and convolutional filters was carried out according to He et al. (2015).

$$\boldsymbol{W}_{Linear} \sim \mathcal{N}(0, \sqrt{\frac{2}{fan_{in}}}) \tag{3.22}$$

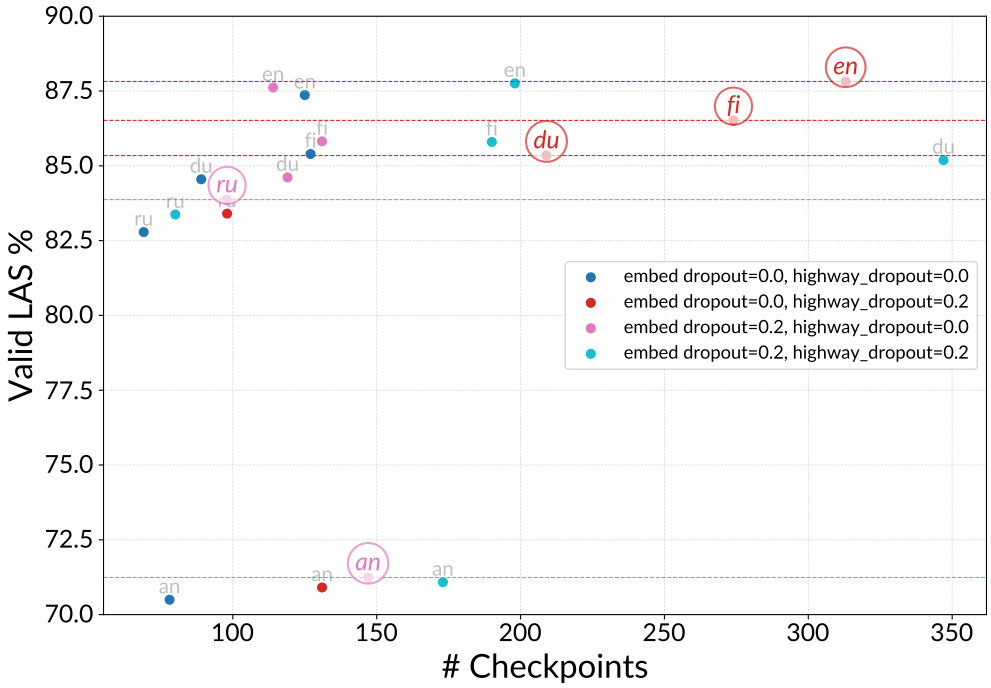


Figure 3.6: Effect of dropout on char-CNN Labelled Attachment Score. The best resulting experiment for each language is circled. Embed dropout is dropout applied to the concatenated word and part of speech embeddings. Highway dropout is dropout applied to the input of the highway layer. Languages are from top to bottom: English, Finnish, Dutch, Russian and Ancient Greek.

where fan_{in} is the dimensionality of the input. All biases were initialised to $\mathbf{0}$. The 8 LSTM weight matrices were initialised with samples from a Gaussian distribution.

$$\boldsymbol{W}_{LSTM} \sim \mathcal{N}(0, \sqrt{\frac{1}{fan_{in}}}) \tag{3.23}$$

Lastly, word embeddings were initialised with samples from a Gaussian distribution with mean 0 and standard deviation 1.

$$\boldsymbol{W}_{embed} \sim \mathcal{N}(0, 1) \tag{3.24}$$

3.2.5.2 Optimiser

We used Adam (Kingma and Ba, 2014) with the default setup (learning rate=0.001, $beta_1$=0.9, $beta_2$=0.999) as our optimiser. We did so to further narrow down our search space since if we had chosen to use stochastic gradient descent we would need to fine tune the learning rate and the learning rate decay. However, we

	word-BILSTM	char-BILSTM	char-CNN
arc dropout	0.2	0.2	0.2
label dropout	0.6	0.6	0.6
sent BILSTM dropout	0.6	0.6	0.6
char BILSTM dropout	—	0.6	—
word level dropout	0.4	0.2	0.0
char embedding dropout	—	0.2	—
highway dropout	—	—	0.2

Table 3.5: Choice of dropout for our models.

are aware that results may have been better if we used stochastic gradient descent, since there has been recent work suggesting that adaptive training methods fail to generalise as well (Wilson et al., 2017).

3.2.5.3 Training Schedule

We trained using mini-batches, the size of which is tabulated in Table 3.4. At each epoch we shuffled the training set and split the sentences into buckets of width 5. Namely, a sentence with length 7 could be in the same bucket with a sentence of length 5 or 9. Shorter sentences are padded and the resulting extraneous activations are not taken into account.

We trained our word-BILSTM and char-BILSTM models performing checkpoints every 100 batches. For our char-CNN we performed a checkpoint every 200 batches. We did so to keep the interval of the checkpoints equal, given that we used a batch size of 16 instead of 32 for the char-CNN. We stopped training when the LAS on the development set didn't improve for 50 consecutive checkpoints.

Our implementation is available at https://github.com/andreasgrv/johnny. Our setup for each model is stored as a yaml configuration file which we have also included in Appendix I. We note that training times varied depending mostly on dataset sizes. Training the char-BILSTM was clearly the slowest. When limiting our training to stop after 50 checkpoints with no improvement, it took 68 hours to train a model for Czech (albeit with approximately 1.5% gains on LAS and UAS after the first day). Other languages converged much faster, some in a few hours

with medium sized ones such as English taking a bit less than a day. All training was carried out on CPU. Example training times per epoch can be seen in Table 3.6.

	word-BILSTM	char-BILSTM	char-CNN
English	240 seconds	455 seconds	310 seconds

Table 3.6: Average training time per **epoch** for English ($\approx$ 12500 sentences). The experiments were carried out on CPU.

4 Results

In this section we report results for our neural graph based dependency parser with a word encoder (word-BILSTM), a character Bidirectional Long Short Term Memory (char-BILSTM) encoder and a character Convolutional Neural Network (char-CNN) encoder. We note that the hyperparameters and training settings employed here are those found to be effective in the previous section on the Universal Dependencies v1.3 development set, and will only be elaborated on when necessary. For an introduction to the datasets and details on how we split our training data for the CONLL 2006 dataset, see Section 2.3.1.

After presenting our results we will examine the outputs of our models and carry out a qualitative analysis for cherry picked input for Ancient Greek.

4.1 Quantitative analysis

4.1.1 CONLL 2006

In this section, we show that:

1. Our subword models outperform our word model on all three languages (Czech, German, Turkish)

2. Our models are competitive on the CONLL 2006 dataset, outperforming our reference model (Zhang et al., 2016).

We begin by discussing our results on the CONLL 2006 dataset. Statistics for this dataset can be seen in Table 4.1. We note that all languages have a sizeable amount of non-projective sentences and that the Czech and German datasets are much larger than the Turkish dataset. We only evaluated on three of the languages in the CONLL 2006 dataset since our original goal was to compare to Zhang et al. (2016).

Language	# sentences	% projective
Turkish	4747	88.29
Czech	69067	76.84 †
German	37255	72.31 †

† Slight discrepancies in numbers from Zhang et al. are due to our different train-dev split.

Table 4.1: CONLL 2006 dataset statistics.

In contrast to Cheng et al. (2016), we did not fine-tune any parameters on the CONLL 2006 dataset. We used the Chu-Liu-Edmonds algorithm to decode the sentences for all languages, since all of them have a reasonably high percentage of non-projectivity. For evaluation we follow the guidelines set out by all other submissions and use the gold part of speech tags at test time. We then score our models by running our output conll file through the **eval.pl** script which doesn't take into account punctation when scoring the sentences.

In Table 4.2 a comparison of our three models to related work can be seen. Word level models are denoted by *w* while character level models by *c*. All models are neural models apart from (Nivre et al., 2006).

To begin with, we note that our char-BILSTM and char-CNN outperform word-BILSTM on all languages, especially Turkish where the char-CNN improves upon the word-BILSTM Labelled Attachment Score by nearly 7%. Moreover, we note that our word-BILSTM improves on reported results by Zhang et al. (2016) mostly on Labelled Attachment Score by about 1% for both Czech and German. We attribute this to the fact that we jointly train our model to predict heads

	Czech		German		Turkish	
	UAS	LAS	UAS	LAS	UAS	LAS
Nivre et al. (2006)[w]	84.80	78.42	88.76	85.82	75.82	65.68
Ballesteros et al. (2015)[c,n]	—	—	—	—	76.32	64.34
Zhang et al. (2016)[w,n]	89.68	81.72	92.19	89.60	—	—
Cheng et al. (2016)[w, n]	**91.16**	**85.14**	92.71	89.90	**78.43**	**66.16**
word-BILSTM[w, n]	89.14	82.88	93.31	90.85	74.85	58.47
char-BILSTM[c, n]	90.48	85.06	**93.63**	**91.73**	76.74	64.53
char-CNN[c, n]	90.26	84.76	93.17	91.15	78.25	65.17

[n] Neural model. [w] Word model. [c] Character model.

Table 4.2: Labelled and Unlabelled Attachment Scores on the CONLL2006 dataset. Best UAS and LAS per language is highlighted in **bold**.

and labels, while their model first predicts heads using their parser network and then predicts labels using a separate network. These results are encouraging, especially given that our model is using a limited vocabulary size of 5000 and has less BILSTM units and a smaller embedding size (we use 200 for both while they use 300).

According to Cheng et al. (2016), their results for Czech, German and Turkish are state of the art on the CONLL 2006 dataset. Overall we notice that their system outperforms ours for Czech but all our models for German outperform theirs—even our word-BILSTM. This is interesting, as their model apart from part of speech tags also employs lemmas and morphological annotations when they are available, albeit combining them using addition instead of concatenation. Nevertheless, as we discussed in Section 2.2.4, their model is very similar to that of Zhang et al.. To get further insight about how the two models compare, we also inspected their performance on English and Chinese, since they both evaluated on the PTB and CTB datasets. We found that Zhang et al. outperforms Cheng et al. on LAS for both languages. We hypothesise that one of the reasons their model outperforms Zhang et al. on the CONLL 2006 is because among other differences, they fine tuned the hidden unit size for each language. They use a unit size of 224 for Czech

and 200 for German which is almost the same as our setup—we use 200 for both languages.

Lastly, for Turkish we note that the non neural model of Nivre et al. (2006) is reasonably effective compared to more recent neural approaches. As we saw earlier in Table 4.1, the Turkish dataset is significantly smaller than the Czech and German datasets. This hints that while neural network approaches can be very competitive on large datasets, they fall short on small datasets where careful manual feature engineering and simpler linear models can go a long way.

4.1.2 Universal Dependencies v1.3

In this section we describe the two sets of experiments we carried out on the Universal Dependencies v1.3 dataset. We compare our word-BILSTM model to the two subword models on 12 languages using word embeddings and gold part of speech tags as we had done for the CONLL 2006 dataset. For the second part of the experiments, we compare a modified version of our char-BILSTM model to other systems on the Universal Dependencies v1.3 dataset. For this modified version we don't use part of speech tags in any way, to make comparison to other systems that use predicted part of speech tags fair. As we shall see, this also serves as an ablative study for our char-BILSTM model, measuring how much its performance depends on part of speech tag input. The most important conclusion from our experiments in this section is that:

1. **The improvements gained by our subword models over our word model are mostly significant for languages that have a high out of vocabulary rate.**

We also show that:

1. It is important to use a decoding algorithm that doesn't enforce projectivity for languages that are highly non-projective.

2. The char-BILSTM model is significantly less effective when used without part of speech tag information.

3. Even without part of speech information, the char-BILSTM obtains sensible results on this dataset compared to related work.

	dev OOV %	type to token %	# sentences	projective %
Finnish	44.77	30.21	12217	92.40
Russian	44.01	33.45	4029	89.23
Turkish	41.18	35.53	3948	84.80
Polish	39.55	32.51	6800	99.53
German	36.72	18.34	14118	88.79
Czech	35.51	10.68	68495	87.88
Ancient Greek	35.13	20.80	13185	37.20
Indonesian	33.82	19.71	4477	98.12
Greek	26.74	21.76	1929	72.11
Dutch	25.82	14.11	13000	71.30
French	25.20	11.85	14554	87.06
English	24.62	9.61	12543	94.78
Arabic	18.85	10.28	6174	90.18
Hebrew	17.26	12.24	5241	100.00

Table 4.3: Languages sorted by percentage of out of vocabulary words (OOV) for the Universal Dependencies v1.3 datasets. The out of vocabulary percentage is calculated on the development set based on the 5000 most common word vocabulary constructed on the training set

4. Our subword models significantly outperform our word model on most languages in this dataset as well.

5. Our char-BILSTM slightly outperforms our char-CNN model, but not by a great margin.

In Table 4.3, statistics on the training sets for languages in the Universal Dependencies v1.3 can be seen. The languages are sorted by out of vocabulary rate on the development set. As noted earlier, we use a fixed word vocabulary size of 5000 for all languages.

Before analysing our results, in Table 4.4 we demonstrate the effect of decoding with the Chu-Liu-Edmonds algorithm or the Eisner algorithm for the most and least projective languages in our dataset. As can be seen, if we use Chu-Liu-

	projectivity %	Chu-Liu-Edmonds		Eisner	
		UAS	LAS	UAS	LAS
Hebrew	100.00	89.22	84.88	89.14	84.80
Ancient Greek	37.20	80.01	71.53	76.88	68.67

Table 4.4: Effect of choice of decoding algorithm on UAS and LAS of most projective and most non-projective language. As can be seen decoding a highly non-projective language, such as Ancient Greek, using the Eisner algorithm (enforces projectivity) leads to significantly lower UAS and LAS scores. The results are for the development set of the Universal Dependencies v1.3 dataset.

Edmonds to decode the output of the parser for Hebrew that has only projective sentences, there is no marked difference in the performance. If on the other hand we use the Eisner algorithm and constrain the output to be projective for a language that is highly non-projective such as Ancient Greek, we get a drop in performance of about 3% in both UAS and LAS. This is evidence that these models generally have a bias towards forming the right output for the majority of cases. The decoding algorithms are used mostly to enforce correctness (no cycles) and don't have much stake in the outcome(Zhang et al., 2016; Cheng et al., 2016; Chorowski et al., 2016).

4.1.2.1 Comparison of word-BILSTM to char-BILSTM and char-CNN

The results for our three models on the Universal Dependencies v1.3 dataset are tabulated in Table 4.5. We see that overall the word-BILSTM model is outperformed for all languages apart from Indonesian and Dutch. For the case of Dutch we noticed a big discrepancy between the scores we obtained on the development set and those we obtained on the test set. However, other submissions Alberti et al. (2017) also had scores in the same ballpark for Dutch. Furthermore, the char-BILSTM seems to generally outperform the char-CNN but not by a great margin—the greatest differences being for Polish and Russian. This is in contrast to Yu and Vu (2017) who found that their CNN subword model performed much better than their implementation of the character BILSTM model when used with their transition parser.

	word-BILSTM		char-BILSTM		char-CNN	
	UAS	LAS	UAS	LAS	UAS	LAS
Hebrew	88.17	83.97	88.35	84.16	**88.42**	**84.29**
Polish	90.27	85.14	**93.26**	**89.34**	92.44	88.03
Indonesian	**84.90**	80.22	84.64	80.14	84.82	**80.51**
English	89.23	86.88	89.58	87.05	**89.70**	**87.25**
Finnish	80.24	75.50	**87.22**	**85.47**	87.12	85.30
Arabic	83.06	78.44	**84.51**	**80.08**	84.31	79.85
Russian	81.95	77.74	**85.68**	**82.37**	84.73	80.99
Czech	89.54	85.62	**91.41**	**88.49**	90.88	87.78
Turkish	82.32	70.26	**85.11**	**77.63**	84.67	77.07
Greek	85.09	82.04	85.83	83.19	**86.20**	**83.27**
Dutch	83.26	**79.80**	**83.48**	79.72	83.31	79.45
Ancient Greek	71.08	64.96	**79.16**	**73.36**	78.66	72.84

Table 4.5: Labelled and Unlabelled Attachment Scores on the Universal Dependencies v1.3 dataset for our word model(word-BILSTM) and two subword models (char-BILSTM, char-CNN). The first group of languages is post processed using the Eisner algorithm while the second is post processed using the Chu-Liu-Edmonds algorithm. The split is carried out according to the percentage projective sentences in the datasets, the threshold being 90%.

	dev OOV %	best char - word	
		UAS	LAS
Finnish	44.77	6.98	9.96
Russian	44.01	3.73	4.63
Turkish	41.18	2.79	7.36
Polish	39.55	2.99	4.20
Czech	35.51	1.87	2.87
Ancient Greek	35.13	8.08	8.40
Indonesian	33.82	-0.08	0.29
Greek	26.74	1.11	1.23
Dutch	25.82	0.22	-0.08
English	24.62	0.47	0.37
Arabic	18.85	1.45	1.64
Hebrew	17.26	0.25	0.32

Table 4.6: Comparison of differences in Unlabelled Attachment Score and Labelled Attachment Score between the best subword model and word-BILSTM on the Universal Dependencies v1.3 dataset. Languages are sorted according to out of vocabulary percentage on the development set. For all languages we used a fixed size vocabulary of 5000, so the size of the datasets also affects the OOV rates.

In order to more clearly see the improvements of the char-BILSTM and char-CNN over the word-BILSTM, we tabulated the difference in LAS and UAS between the best performing subword model and the word-BILSTM. We then ordered the languages according to out of vocabulary rate on the development set because an interesting pattern emerges. As can be seen in Table 4.6, our results seem to agree with the findings of Yu and Vu. They highlight that the improvement of the subword models comes mostly from their ability to solve the issues word level models have due to their finite vocabulary size. As an example, for Finnish with an out of vocabulary rate of 44.77% on the development set, the improvements in UAS and LAS are 6.98 and 9.96 respectively while for Hebrew with the lowest out of vocabulary percentage the improvements are very modest. This suggests that although the subword models can provide

substantial improvements for morphologically-rich languages, it is mostly due to the subword models being able to come up with a better constructed embedding than the unknown embedding word level models fall back on. These constructed embeddings may capture some general morphological characteristics in order to make these embeddings optimal, but they are not required to. All that is required is that the representations minimize the loss, and in our case the training loss only takes into account predicting correct arcs and labels;there is no loss related to part of speech tagging or morphological annotation to drive the character embeddings to necessarily capture explicit morphological patterns.

We notice that our subword level models didn't perform well on Indonesian, Hebrew and Arabic. Hebrew and Arabic are classified as root and pattern languages, while Indonesian's morphology presents a lot of reduplication. On the other hand, we do get large improvements in Finnish and Turkish that are agglutinative. This may be a more general pattern, since agglutinative languages have clearly separated morphs as we discussed in Section 2.1.1. However, we didn't notice any trends regarding CNNs outperforming LSTMs on agglutinative languages as reported in the literature (Yu and Vu, 2017). Furthermore, we also get large improvements for Ancient Greek which is fusional. Ancient Greek has very loose word order and is highly non-projective, further investigation is needed to see if this is the source for the drastic improvement.

4.1.2.2 Comparison to related work using modified char-BILSTM

In this subsection we compare our char-BILSTM model to other parsers on the Universal Dependencies v1.3 dataset. We modify our char-BILSTM model to not use part of speech tags in the input. We do so to make the comparison to the other systems that use predicted part of speech tags fair.

As we discussed in the previous section, we didn't fine tune on a particular language, in contrast to Chorowski et al. (2016) who fine tuned their graph based parser on Polish. Their model is similar to our char-CNN but is much larger. They use 1050 filters projected to 512 units that are transformed with 3 highway layers. That is followed by 2 layers of GRUs with 512 units. Our BILSTM has 200 units, and even though we compose using concatenation instead of addition and have LSTMs instead of GRUs, the number of parameters in our model is much smaller.

Alberti et al. (2017) is a transition based parser that is jointly trained to predict part of speech tags and parsing actions. Similar to our subword models it is a

character model. At decoding time they use a beam of width 8. As can be seen in Table 4.7, our model is still competitive given its size and the fact that we do not use part of speech tags in any way.

We note that for Ancient Greek the char-BILSTM and the model of Chorowski et al. significantly outperforms the transition based parser. We attribute this to the high degree of non-projectivity the Ancient Greek treebank has.

	Alberti et al.$^{\diamond}$		Chorowski et al.		char-BILSTM†	
	UAS	LAS	UAS	LAS	UAS	LAS
Czech	89.09	84.99	**91.41**	**88.18**	90.39	86.77
Polish	**91.86**	**87.49**	90.26	85.32	90.33	84.48
Russian	**84.27**	**80.65**	83.29	79.22	82.95	78.46
German	**84.12**	**79.05**	82.67	76.51	83.26	77.56
English	**87.86**	**84.45**	87.44	83.94	87.11	83.06
French	86.61	83.10	**87.25**	**83.50**	87.08	83.19
Ancient Greek	73.85	68.10	**78.96**	**72.36**	77.28	69.92

$^{\diamond}$ This is a transition based parser.

† Our modified char-BILSTM that doesn't use part of speech tags.

Table 4.7: Comparison of Labelled and Unlabelled Attachment Scores on the Universal Dependencies v1.3 dataset for our word model char-BILSTM modified to not use part of speech tags.

As promised, we now demonstrate the effect of using part of speech embeddings on the input for the char-BILSTM model. As can be seen in Table 4.8, if we compare our modified char-BILSTM model to the original one that leverages gold part of speech tags, we see that its performance has dropped dramatically. We note that UAS is affected less than LAS with a drop of 2-3%. The drop in LAS is generally approximately 3-4%, although Polish has a nearly 5% drop in LAS! An important note however, is that for Czech—for which we have a much larger dataset—the drop is much less serious being about 1% UAS and approximately 2% LAS. This suggests that it may be the case that the subword models can learn interesting patterns without guidance, but the number of training examples needed is much greater.

While this decline in attachment scores could in part be attributed to the fact

	char-BILSTM + POS		char-BILSTM - POS	
	UAS	LAS	UAS	LAS
Czech	**91.41**	**88.49**	90.39	86.77
Polish	**93.26**	**89.34**	90.33	84.48
Russian	**85.68**	**82.37**	82.95	78.46
German	—	—	83.26	77.56
English	**89.58**	**87.05**	87.11	83.06
French	—	—	87.08	83.19
Ancient Greek	**79.16**	**73.36**	77.28	69.92

Table 4.8: Illustration of drop in performance for char-BILSTM model when part of speech embeddings are not used for Universal Dependencies v1.3 dataset. Best UAS and LAS per language is highlighted in bold.

that we didn't fine tune any parameters for the model with no part of speech tags, we believe this factor alone is clearly not enough to explain such deviation. To summarise, while character level models can create good representations for words, the training signal sources are an important factor. As may be obvious, we cannot treat subword models as black boxes expecting that they will automatically capture the patterns we want, without explicitly guiding them. Both Alberti et al. and Chorowski et al. use part of speech tag prediction for their training signal apart from arc and label prediction for parsing. And even under such guidance, it is still unclear to us if such models will capture what we humans believe to be intuitive. In the next section, we shall briefly touch on this subject by explicitly presenting the internal state of the network for some cherry-picked Ancient Greek sentences.

4.2 Qualitative analysis

In this section, we shall analyse whether our word-BILSTM and char-BILSTM models[1] seem to be able to capture the case of nouns in Ancient Greek. As we discussed in Section 2.1.1, in Ancient Greek the subject of a verb is in the nominative case, while the object is usually in the accusative or the dative.

[1] We use the models which also leverage part of speech tags in the input.

We begin this analysis with a sentence from the training set. We will only keep track of the case of the nouns in the first row of the glosses, since this is what matters for our current analysis.

(4.1) καὶ γὰρ ὁ νομοθέτης Σόλων ἔφη .
nom nom nom
And so lawmaker Solon said .
"And so lawmaker Solon said"

In Figure 4.1a, the correct parse for this sentence is illustrated. In the matrices below, the input sequence is represented along the x axis. Each column illustrates a probability distribution. Dark blue cells represent values near 1 and lightly shaded cells values near 0. The prediction the model makes can be seen on the y axis of the matrices.

The top two matrices show the probability distribution over the labels for each input word. As an example, we see that "Σόλων" is predicted with high probability to be a nominal subject (*nsubj*) under the char-BILSTM, but for the word-BILSTM there is also some probabilty mass on direct object (*dobj*). Note that we have removed labels that didn't have any probability mass on them over the whole sentence for clarity—if we rendered all of them, the matrix would be very tall.

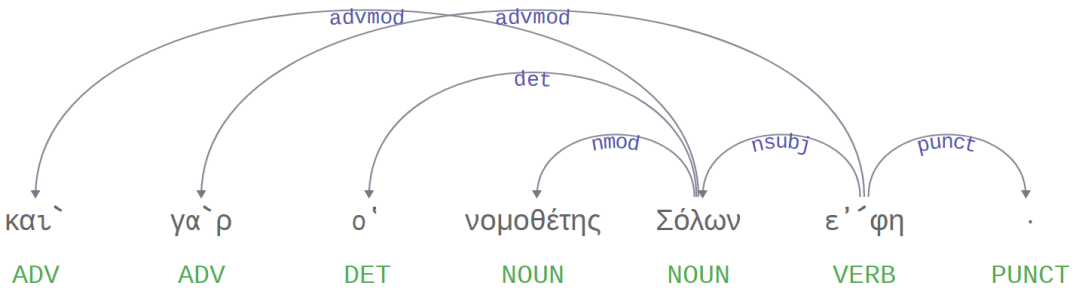
In the bottom two matrices, the probability distribution over likely heads is displayed. The y axis has $n + 1$ labels since for each of the n input words we can predict that the head of the word is our favourite artificial token, the <*ROOT*>.

We note that the differences between the two systems on the training set are minor. Both have very peaked probability distributions for this example, with the exception of the choice of head for "καὶ" which is generally ambiguous, since it is also used as a conjunction.

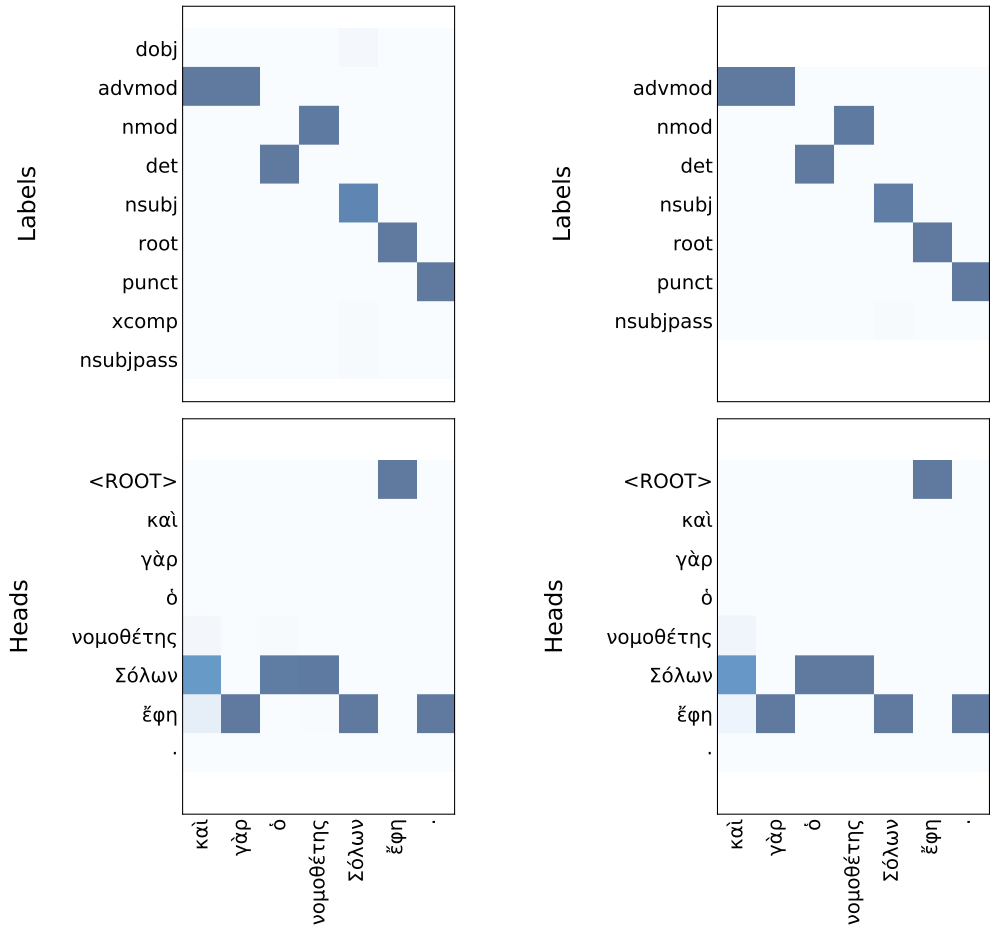
4.2.1 Do our models capture patterns of case inflection?

We shall now modify this example to see how the models extrapolate to slight deviations from the training set. We add "τῷ Λυσάνδρῳ" (determiner + Lysander in dative case) before the verb "ἔφη".

(4.2) καὶ γὰρ ὁ νομοθέτης Σόλων τῷ Λυσάνδρῳ ἔφη .
nom nom nom dat dat
And so lawmaker Solon to Lysander said .
"And so lawmaker Solon said to Lysander ."



(a) Correct parse - predicted correctly by both models.



(b) word-BILSTM **(c)** char-BILSTM

Figure 4.1: Internal state of word-BILSTM and char-BILSTM models on sentence from the training set. Both models predict the sentence correctly (for example the first column shows that the predicted head of "καὶ" is "Σόλων" with a label of "advmod". The corresponding dependency graph can be seen above.

The results for this sentence are illustrated in Figure 4.2. We emphasise that for the word model, we have replaced the part of speech tags in green below the words with the word itself if it is in vocabulary and *UNK* otherwise. As can be seen, the word-BILSTM attaches "Λυσάνδρῳ" as modifying "Σόλων". This behaviour would make sense if we had inflected Lysander to be in the genitive—not the dative. We justify this error by noting that "Λυσάνδρῳ" is out of vocabulary; the word model is mostly using the part of speech to make this decision. On the other hand, the char-BILSTM attaches "Λυσάνδρῳ" as the object of the verb, which is what we would expect in this case. Nevertheless, we note that the char-BILSTM also shows a level of activation for the noun modifier label.

4.2.2 Do our models capture case agreement?

We now manipulate the order of the sentence further, to see if the models are learning to attach words that agree in case. The following may not be a valid Ancient Greek sentence, but it is interesting to see whether the model is capturing the inflections of words or mostly using proximity in the sentence to draw the arcs.

(4.3) καὶ γὰρ ὁ νομοθέτης τῷ Σόλωνι Λύσανδρος ἔφη .
nom nom dat dat nom
And so lawmaker to Solon Lysander said .
"And so lawmaker Lysander said to Solon ."

This time we inflect Solon to be in the dative and leave "Λύσανδρος" in the nominative. Note that "νομοθέτης" is still in the nominative and closer in the sentence to "Σόλωνι", that is in the dative case. As we see in Figure 4.3, the word-BILSTM sets Lysander that is in the nominative as an object, which is not reasonable. Amusingly, we note that the char-BILSTM does what would we would expect, it attaches "Λύσανδρος" to "νομοθέτης" which agree in case and sets "Λύσανδρος" to be the subject.

We conclude this analysis with our familiar example from Aesop's fables.

(4.4) Τοῦτο ἀλώπηξ κατιδοῦσα ἐβόα .
acc nom nom .
This fox saw cried aloud .
"When the fox saw this it cried aloud ."

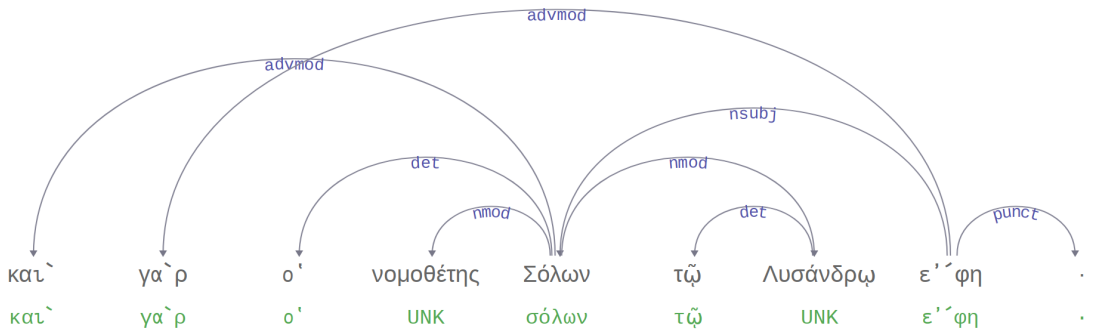
This sentence is from the development set. As can be seen in Figure 4.4, both system outputs are erroneous, since they attach "Τοῦτο" (this) to "ἐβόα" instead

of the participle “κατιδοῦσα”. However, the word-BILSTM model additionally assigns a nominative cased noun “ἀλώπηξ” as an object, while the char-BILSTM correctly assigns it as a subject.

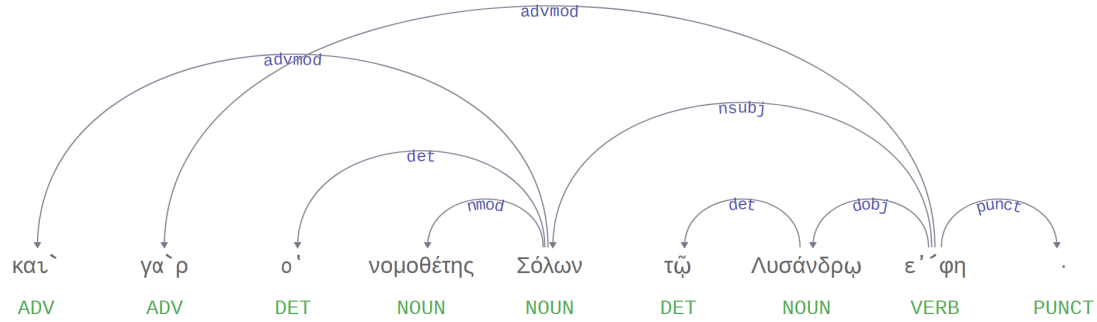
To summarise this analysis, we saw that the char-BILSTM word representation seems to embody much more useful information than the word level model. We demonstrated that there is evidence that the char-BILSTM may be capturing information about case. However, the extent to which this is true for a large number of unseen sentences and how well it generalises to larger ones is a question we shall defer for future work, as is the influence of using part of speech information.

We note at this point that there are a few decisions that may have biased the analysis we have carried out here in favour of the char-BILSTM.

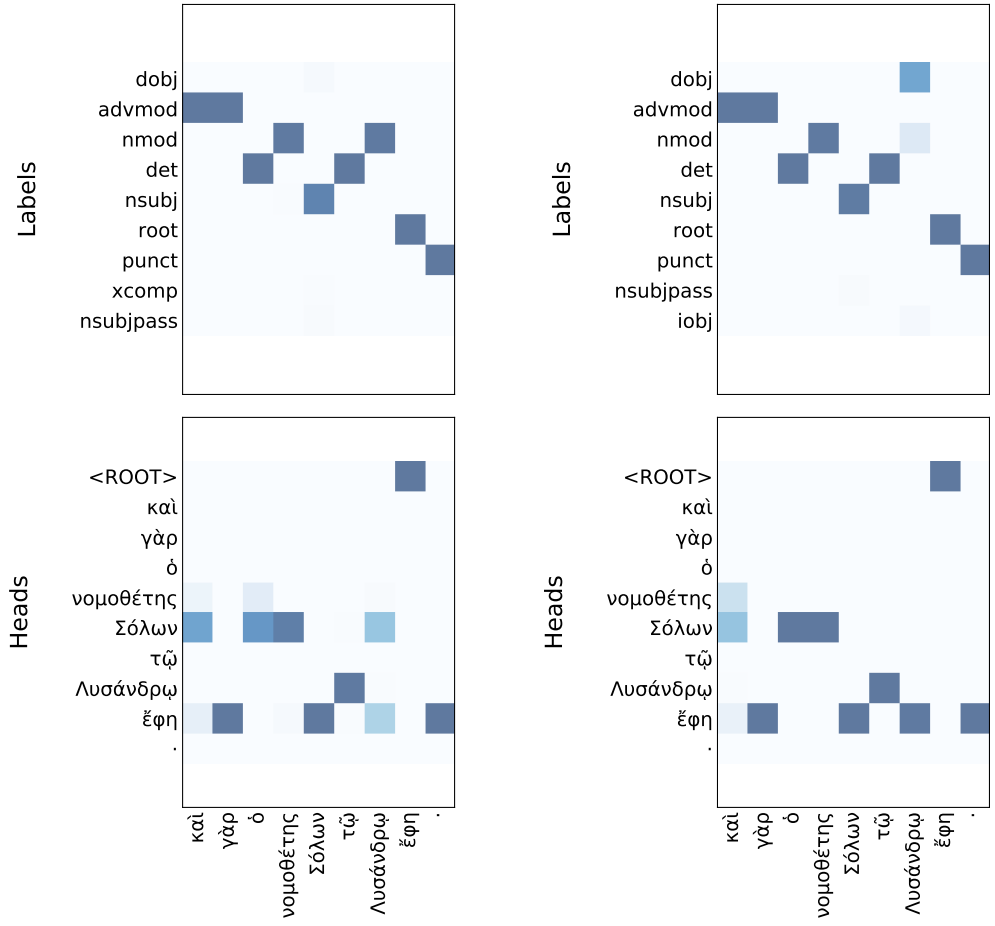
- The sentences we chose had many words that were out of vocabulary (although this may be unavoidable with a 5000 word vocabulary).
- three of the sentences were modified versions of a sentence that was in the training set.
- The validity of the sentences in Ancient Greek needs to be ascertained.



(a) Word-BILSTM parse

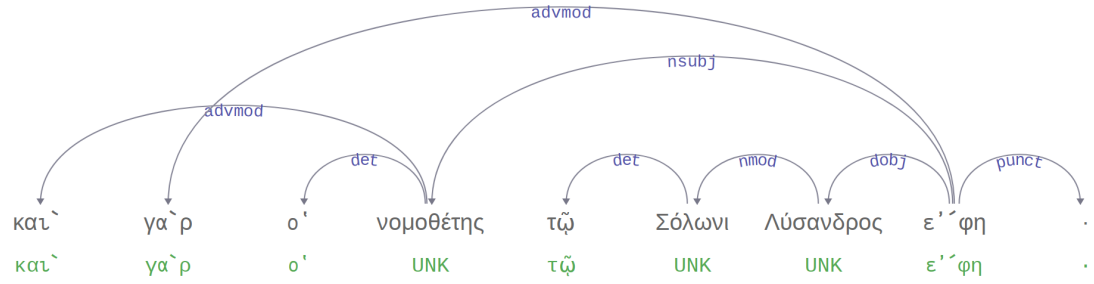


(b) Char-BILSTM parse

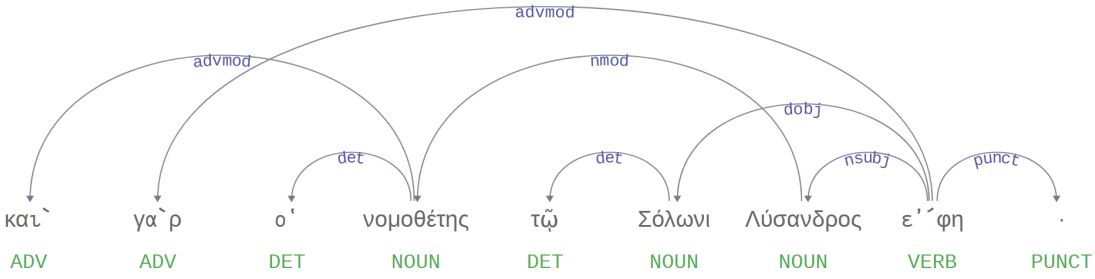


(c) word-BILSTM (d) char-BILSTM

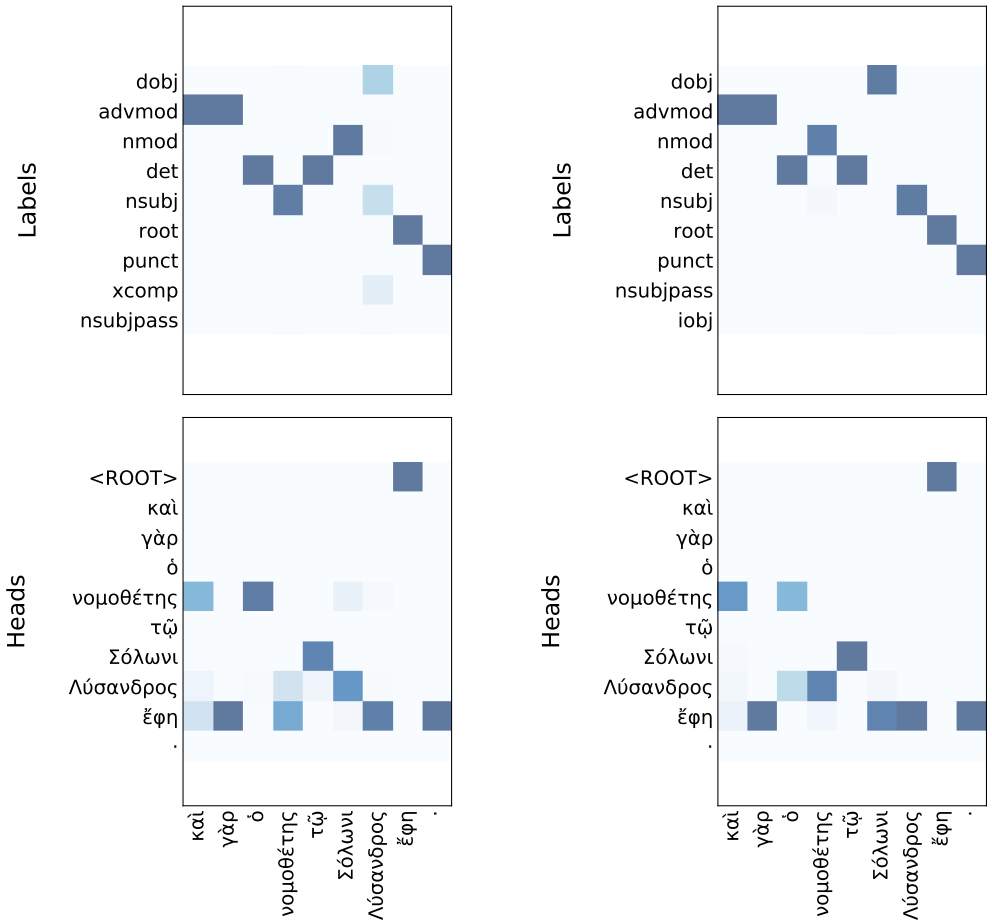
Figure 4.2



(a) Word-BILSTM parse

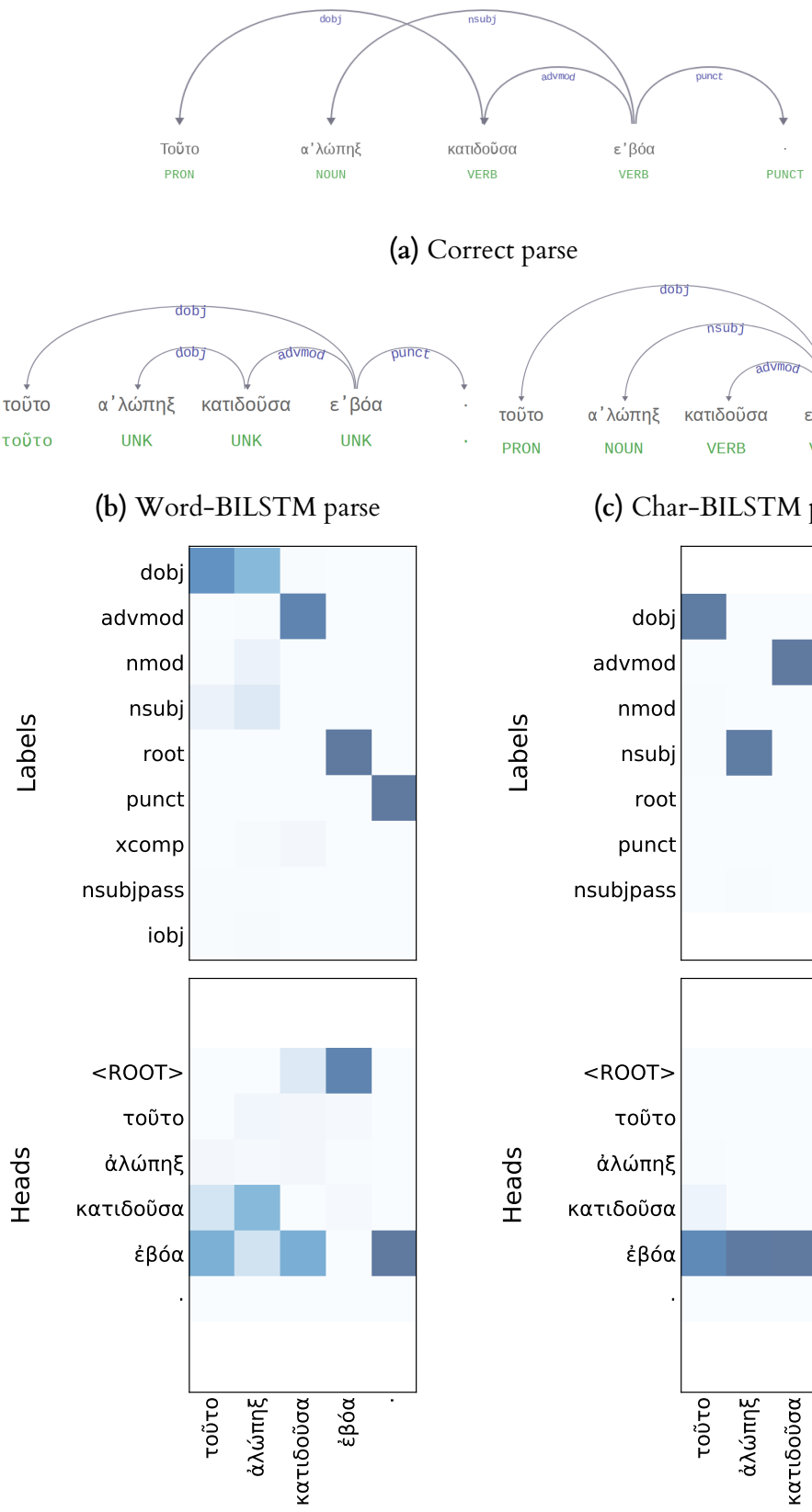


(b) Char-BILSTM parse



(c) word-BILSTM (d) char-BILSTM

Figure 4.3



(a) Correct parse

(b) Word-BILSTM parse

(c) Char-BILSTM parse

(d) word-BILSTM

(e) char-BILSTM

Figure 4.4: Comparison of word-BILSTM to char-BILSTM internal state for sentence of development set. As can be discerned from the correct parse, both system outputs are erroneous.

5

Conclusions & Future work

We argued that the char-BILSTM and char-CNN subword level encoders we explored are significantly more effective than the word-BILSTM encoder for parsing morphologically-rich languages. We have demonstrated that this is so, first and foremost because the subword encoders mitigate the out of vocabulary issues that word level models have. Subword level models distribute the weights more effectively, as they place the parameters over tokens that are guaranteed to be used. This is in contrast to word level models where the embeddings of some rare words can be thought of as being a bad investment of parameters, since not only will they be used for a very small percentage of inputs, there will also not be many training examples to train them on.

The char-BILSTM was found to slightly outperform the char-CNN on most languages. However, the char-CNN model is more compelling computationally, as we found convolutional networks faster to train, while at the same time being more parallelisable than recurrent ones. We demonstrated that apart from better performance, the training curves for the subword level encoders were also more well-behaved than the word level encoders for morphologically-rich languages.

We note that we haven't clearly answered whether morphological analysis is an important prerequisite for parsing morphologically-rich languages. While in the previous section we demonstrated that the char-BILSTM model seems to be

capable of detecting patterns of case inflection in Ancient Greek nouns, it is still unclear to what extent this is true. We certainly cannot be confident enough to conclude much from these limited initial experiments, despite the attention they seek. Furthermore, given the sharp decline in attachment scores we witnessed for the char-BILSTM without part of speech embeddings, it seems fairly clear that subword models cannot magically come up with good representations without explicit guidance. This is true at least for dependency parsing for which datasets are tiny compared to other tasks, such as language modelling. We therefore expect that morphological analysis information would help improve the model further. Nevertheless, we shun jumping to conclusions with little evidence and look forward to investigating this further. We plan to do so by exploring ways of applying attention on the character level to better probe what the model is learning in its character representations.

Another obvious route for future work is to use part of speech information more effectively. This can be done by predicting tags instead of providing them as input. As we saw, other approaches report improvements when using part of speech tags this way. Moreover, the model is clearly more compelling, since we no longer need to find a part of speech tagger to generate the tags for input at test time. Along this line of work, we can also come up with ways to incorporate lemmas and morphological annotations into our loss function. This way, we would bias the word embeddings towards containing information that we believe would be advantageous for our model.

Finally, a more rigorous analysis and approach for combining the losses incurred by the parser is in order. Our method following Kiperwasser and Goldberg (2016) simply adds the losses from the arc prediction and arc labelling tasks. However, we believe this to be a bit naive. For one there is no reason to believe that these two losses should be weighted the same. This is so, since their respective output vocabularies are different and in general larger output vocabularies are expected to have larger losses. Chorowski et al. (2016) weight the losses incurred from part of speech tagging, arc prediction and label prediction, but to our knowledge they choose the weight coefficients without justification. On a final note, related work (Alberti et al., 2017) also uses a complex switching mechanism between tasks during training, which suggests that there is ample room for experimentation and improvement[1].

[1] hopefully.

Part I

Appendix

Label	Description
acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list
mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement

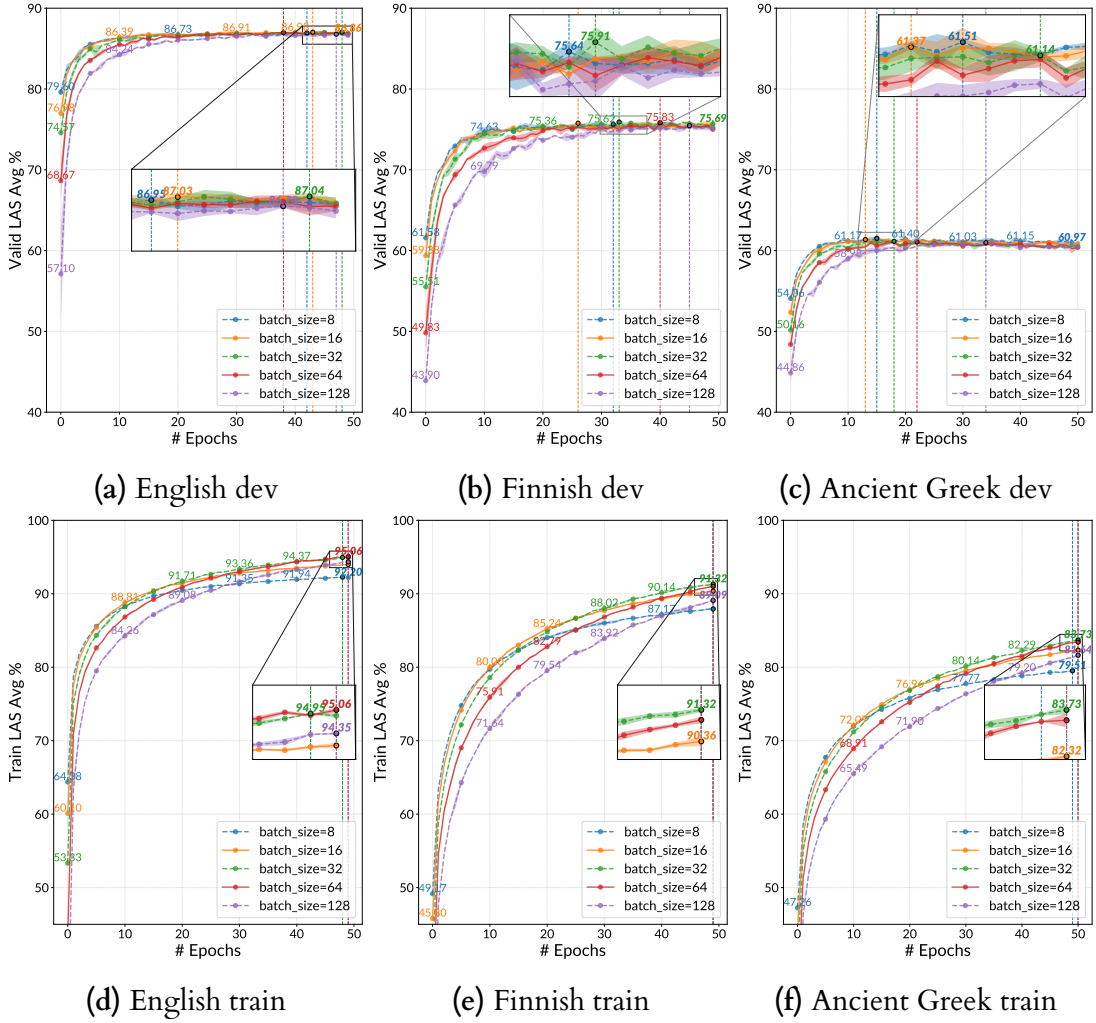
Table 1: Explanation of 37 dependency labels used in Universal Dependencies v2.0

```yaml
seed: 43
ngram: 0
max_epochs: 1000
batch_size: 32
dev_batch_size: 256
vocab:
  size: 5000
  threshold: 0
dataset:
  name: CONLL2017_v2_0
  lang: English
model:
  module: johnny.models
  classname: GraphParser
  encoder:
    module: johnny.components
    classname: SentenceEncoder
    dropout: 0.6
    embedder:
      module: johnny.components
      classname: Embedder
      dropout: 0.4
      in_sizes:
      - dunno_yet
      - dunno_yet
      out_sizes:
      - 200
      - 30
    num_layers: 2
    num_units: 200
    use_bilstm: true
  mlp_arc_units: 100
  mlp_lbl_units: 100
  arc_dropout: 0.2
  lbl_dropout: 0.6
  num_labels: dunno_yet
  treeify: none
optimizer:
  grad_clip: 5
  learning_rate: 0.001
preprocess:
  collapse_nums: true
  collapse_triples: true
  lowercase: true
  expand_diacritics: false
  remove_diacritics: false
train_buckets:
  bucket_width: 5
  right_leak: 5
checkpoint:
  patience: 50
  every: 100
```

```yaml
seed: 43
ngram: 1
max_epochs: 1000
batch_size: 32
dev_batch_size: 256
vocab:
  size: 100
  threshold: 10
dataset:
  name: CONLL2017_v2_0
  lang: English
model:
  module: johnny.models
  classname: GraphParser
  encoder:
    module: johnny.components
    classname: SentenceEncoder
    dropout: 0.6
    embedder:
      module: johnny.components
      classname: SubwordEmbedder
      dropout: 0.2
      word_encoder:
        module: johnny.components
        classname: LSTMWordEncoder
        vocab_size: dunno_yet
        num_units: 200
        num_layers: 1
        rec_dropout: 0.6
        inp_dropout: 0.2
        use_bilstm: True
      in_sizes:
      - dunno_yet
      out_sizes:
      - 30
    num_layers: 2
    num_units: 200
    use_bilstm: true
  mlp_arc_units: 100
  mlp_lbl_units: 100
  arc_dropout: 0.2
  lbl_dropout: 0.6
  num_labels: dunno_yet
  treeify: none
optimizer:
  grad_clip: 5
  learning_rate: 0.001
preprocess:
  collapse_nums: false
  collapse_triples: false
  lowercase: true
  expand_diacritics: true
  remove_diacritics: false
train_buckets:
  bucket_width: 5
  right_leak: 5
checkpoint:
  patience: 50
  every: 100
```

```yaml
seed: 43
ngram: 1
max_epochs: 1000
batch_size: 16
dev_batch_size: 256
vocab:
  size: 100
  threshold: 10
dataset:
  name: CONLL2017_v2_0
  lang: English
model:
  module: johnny.models
  classname: GraphParser
  encoder:
    module: johnny.components
    classname: SentenceEncoder
    dropout: 0.6
    embedder:
      module: johnny.components
      classname: SubwordEmbedder
      dropout: 0.0
      word_encoder:
        module: johnny.components
        classname: CNNWordEncoder
        vocab_size: dunno
        embed_units: 15
        ngrams: [1, 2, 3, 4, 5, 6]
        num_filters: [20, 45, 70, 95, 120,
            145]
        num_highway_layers: 1
        highway_dropout: 0.2
      in_sizes:
      - dunno
      out_sizes:
      - 30
    num_layers: 2
    num_units: 200
    use_bilstm: true
  mlp_arc_units: 100
  mlp_lbl_units: 100
  arc_dropout: 0.2
  lbl_dropout: 0.6
  num_labels: dunno
  treeify: none
optimizer:
  grad_clip: 5
  learning_rate: 0.001
preprocess:
  collapse_nums: false
  collapse_triples: false
  lowercase: true
  expand_diacritics: true
  remove_diacritics: false
train_buckets:
  bucket_width: 5
  right_leak: 5
checkpoint:
  patience: 50
  every: 200
```

Figure 1: YAML configuration blueprints for our three models, word-BILSTM, char-BILSTM and char-CNN respectively.



(a) English dev (b) Finnish dev (c) Ancient Greek dev

(d) English train (e) Finnish train (f) Ancient Greek train

Figure 2: Labelled Attachment Score on the training and development set as a function of batch size over 50 epochs for the word-BILSTM model. This figure illustrates the large differences in LAS for these languages. We note that the word-BILSTM model also fits the data much worse for Finnish and Ancient Greek than for English, due to the fact that the word model isn't as effective for morphologically-rich languages.

Bibliography

Chris Alberti, Daniel Andor, Ivan Bogatyy, Michael Collins, Dan Gillick, Lingpeng Kong, Terry Koo, Ji Ma, Mark Omernick, Slav Petrov, Chayut Thanapirom, Zora Tung, and David Weiss. Syntaxnet models for the conll 2017 shared task. *CoRR*, abs/1703.04929, 2017. URL `http://arxiv.org/abs/1703.04929`.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. One parser, many languages. *CoRR*, abs/1602.01595, 2016. URL `http://arxiv.org/abs/1602.01595`.

Miguel Ballesteros, Chris Dyer, and Noah A Smith. Improved transition-based parsing by modeling characters instead of words with lstms. *arXiv preprint arXiv:1508.00657*, 2015.

Laurie Bauer. Introducing linguistic morphology. 2003.

Bernd Bohnet and Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1455–1465, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=2390948.2391114`.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428, 2013.

Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *In Proc. of CoNLL*, pages 149–164, 2006.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. *CoRR*, abs/1608.02076, 2016. URL `http://arxiv.org/abs/1608.02076`.

Do Kook Choe and Eugene Charniak. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1257`.

Jan Chorowski, Michal Zapotoczny, and Pawel Rychlikowski. Read, tag, and parse all at once, or fully-neural dependency parsing. *CoRR*, abs/1609.03441, 2016. URL `http://arxiv.org/abs/1609.03441`.

Yoeng-Jin Chu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *CoRR*, abs/1611.09913, 2016. URL `http://arxiv.org/abs/1611.09913`.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011. URL `http://arxiv.org/abs/1103.0398`.

Ryan Cotterell and Hinrich Schütze. Morphological word-embeddings. In *HLT-NAACL*, pages 1287–1292, 2015.

Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734, 2016. URL `http://arxiv.org/abs/1611.01734`.

Timothy Dozat, Peng Qi, and Christopher D. Manning. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K/K17/K17-3002.pdf`.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240, 1967.

Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer, 2000.

Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18 (5):602–610, 2005.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple NLP tasks. *CoRR*, abs/1611.01587, 2016. URL `http://arxiv.org/abs/1611.01587`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

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`.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015. URL `http://arxiv.org/abs/1508.06615`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR*, abs/1603.04351, 2016. URL `http://arxiv.org/abs/1603.04351`.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

Marco Kuhlmann and Joakim Nivre. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, COLING-ACL '06, pages 507–514, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1273073.1273139`.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

John Lee, Jason Naradowsky, and David A Smith. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 885–894. Association for Computational Linguistics, 2011.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernández Astudillo, Silvio Amir, Chris Dyer, Alan W. Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *CoRR*, abs/1508.02096, 2015. URL `http://arxiv.org/abs/1508.02096`.

Xuezhe Ma and Eduard H. Hovy. Neural probabilistic model for non-projective MST parsing. *CoRR*, abs/1701.00874, 2017. URL `http://arxiv.org/abs/1701.00874`.

André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural*

Language Processing, pages 34–44. Association for Computational Linguistics, 2010.

André FT Martins, Miguel B Almeida, and Noah A Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. 2013.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

Ryan T McDonald and Joakim Nivre. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*, pages 122–131, 2007.

Dat Quoc Nguyen, Mark Dras, and Mark Johnson. A novel neural network model for joint POS tagging and graph-based dependency parsing. *CoRR*, abs/1705.05952, 2017. URL `http://arxiv.org/abs/1705.05952`.

Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, pages 915–932. sn, 2007.

Joakim Nivre. Universal dependency evaluation. 2016.

Joakim Nivre and Johan Hall. Maltparser: A language-independent system for data-driven dependency parsing. In *In Proc. of the Fourth Workshop on Treebanks and Linguistic Theories*, pages 13–95, 2005.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 221–225, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1596276.1596318`.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In Nicoletta Calzolari (Conference Chair),

Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA). ISBN 978-2-9517408-9-1.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

Milan Straka and Jana Straková. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K/K17/K17-3009.pdf`.

Lucien Tesnière. *Elements of structural syntax*. John Benjamins Publishing Company, 2015.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015. URL `http://learningsys.org/papers/LearningSys_2015_paper_33.pdf`.

Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kübler, Marie Candito, Jennifer Foster, Yannick Versley, Ines Rehbein, and Lamia Tounsi. Statistical parsing of morphologically rich languages (spmrl): What, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, SPMRL '10, pages 1–12, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1868771.1868772`.

Clara Vania and Adam Lopez. From characters to words to in between: Do we capture morphology? *CoRR*, abs/1704.08352, 2017. URL `http://arxiv.org/abs/1704.08352`.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey

Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781, 2015.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *CoRR*, abs/1506.06158, 2015. URL `http://arxiv.org/abs/1506.06158`.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *CoRR*, abs/1705.08292, 2017. URL `http://arxiv.org/abs/1705.08292`.

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

Xiang Yu and Ngoc Thang Vu. Character composition model with convolutional neural networks for dependency parsing on morphologically rich languages. *CoRR*, abs/1705.10814, 2017. URL `http://arxiv.org/abs/1705.10814`.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinkova, Jan Hajic jr., Jaroslava Hlavacova, Václava Kettnerová, Zdenka Uresova, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria depaiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonca, Tatiana Lando, Rattima Nitisaroj, and Josie Li. Conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K/K17/K17-3001.pdf`.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. Dependency parsing as head selection. *CoRR*, abs/1606.01280, 2016. URL `http://arxiv.org/abs/1606.01280`.