Abstract

A classifier is an important component of dependency parsers. The design of relevant features and the amount of training data are major challenges to achieve good parsing results. One aim of this project is to evaluate the performance of two kinds of LSTM parsers (word based and character based) that use simple features with a focus on how the size of the training data influences the results. Another aim is to compare the LSTM parsers with the Stanford Neural Network parser. The UD corpora for English, Swedish and Finnish are used as training, development and test data. I have shown that LSTM parsers with very simple features (coarse POS) have a reasonable performance when trained on UD corpora with limited size. There is clear evidence that the size of the UD English corpora affects the parsing accuracy. That is, the larger the training data, the better results. I have also shown that it is possible to use multi-language corpora as input for training in order to improve the parsing accuracy for the languages with short training corpora. When parsing Swedish UD corpus, the LSTM parsers have a better performance than the Stanford neural network parser.

1 Introduction

There are different principles for handling dependency parsing. Transition-based dependency parsers are popular because they are accurate and they have linear time complexity (Kubler, McDonald, Nivre, & Hirst, 2009). The parser builds the dependency graph for a sentence in steps. At each step the parser has to decide the transition to the next state. The key component is a supervised machine learning classifier that selects the next state (Hall, Nivre, & Nilsson, 2006). There are two issues with supervised machine learning: 1) difficulty to find relevant features and 2) shortage of training data.

Traditionally, a shallow classifier uses a large number of complex features as input. The features are defined manually using extensive expertise in the domain area. For example, in natural language processing grammarians and linguists are often involved in identifying relevant features. This process is called feature engineering; it is based on trial and error and is time consuming. If the classifier is used to process different languages, different features may be needed and experts in the languages must be involved. The manual feature engineering process is a challenge when designing a classifier (Bengio, 2013).

The shortage of training data is a general problem for machine learning. More training data enables more accurate classifiers based on relevant features. Very often the definition of features is based on classifiers that require their own training data. For example, if I need “name entity recognition” as a feature, a corresponding classifier must be used. If I need morphological features a more advanced classifier and more training data is needed. The features must be accurate (large amount of training data and manual verification) because they have a heavy influence on the accuracy of the end-application.

There are methods in machine learning that use simple features as input data and new complex features are automatically defined during the training phase. That is, during the training phase, the error is minimized at the same time that the required features are defined and refined. The manual feature
engineering step is moved into the learning algorithm. Examples are deep neural networks and different kinds of recurrent neural networks. This approach based on simple input features has been successfully used in dependency parsers. For example, (Chen & Manning, 2014) have developed a dependency parser using deep neural network and embedded word representation with good results in English and Chinese. (Dyer, Ballesteros, Ling, Matthews, & Smith, 2015) have developed a parser using stacked recurrent neural network called long short-term memory (LSTM) and embedded word representation with good results in English and Chinese. Another version of LSTM using embedded characters as input has shown good results in a large number of languages (Ballesteros, Dyer, & Smith, 2015).

Currently there are many ways to describe dependency grammar for different languages. This variety of annotations makes it difficult to compare and to re-use dependency parsers across different schemes and languages (McDonald, et al., 2013). Universal dependency (UD) defines a common annotation for describing dependency grammar that can be used by many different languages (Nivre, 2015). One of the goals of UD is to enable the development and evaluation of dependency parsers in different languages.

In order to support the use of Universal Dependencies (UD) into new languages with limited linguistic resources, it is important to have dependency parsers using classifiers with as simple features as possible and requiring a reasonable amount of training data. For instance, new languages may deploy UD in steps with only coarse grained POS tags and no morphological information in the first step.

The aim of this paper is to evaluate the transition-based dependency parsers using LSTM (Ballesteros, Dyer, & Smith, 2015) and (Dyer, Ballesteros, Ling, Matthews, & Smith, 2015) when parsing Swedish, English and Finnish UD data. The LSTM parser uses simple features (words and POS). The study will analyze how the size of training data may impact the LSTM parsers performance.

Another aim is to compare the results of the LSTM parsers with the Stanford Neural Network parser (Chen & Manning, 2014) using Swedish UD corpus.

2 Universal Dependencies

Dependency grammar describes the syntactic structure of a sentence as a set of binary, asymmetric relations between words/tokens.

The principles are to divide the text into sentences. Each sentence is tokenized into syntactic words. Syntactic words (words) are the basic unit when defining dependencies. The words are linked by labelled syntactic relations (labeled dependencies). Function words attach to the most closely related content word where the content word is always the head and the function word is a dependent. UD also defines a set of POS tags and morphological features that are used to describe the words.

CONLL-U format (CoNLL-U format, 2015) is used to describe sentences according to UD.

3 Transition based dependency parser

A transition-based dependency parser builds a dependency graph for a sentence in steps, one word at a time. The basic structure for the parser according to (Nivre, Algorithms for Deterministic Incremental Dependency Parsing, 2008) is a buffer with the tokens in the input sentence, a stack with the remaining tokens, a stack with the partially processed tokens and a set of dependency arcs. The transition-based parses may be classified as arc-standard or arc-eager. The arc-standard algorithm assumes that all dependencies to a word are attached to it before the word itself is attached to its head. Here a bottom up approach is used and the parser may need to go all the way to the end of sentence before attaching a word to its head. In the arc-eager method, a word is attached to its head as early as possible. The following are the actions supported for the arc-standard method: reduce right-arc, reduce left-arc and shift.

4 RNN and LSTM

Recurrent neural network (RNN) is a type of artificial neural network with cycles. RNN can handle sequences of variable lengths, one item at a time where the sequence order is kept. The cycles enable a memory function. That is, the past context may influence the current decisions.

The training of a standard RNN is practically not feasible because the gradients in the backpropagation algorithm tends either to vanish or to explode.
Therefore, the current gradient descent methods used during training do not work.

There are also problems with the memory function when it is passed through many steps/states; it becomes too diluted and it loses its content.

### 4.1 LSTM

LSTM (Long Short-Term Memory) is a type of RNN that resolves the issues of limited memory and vanishing gradients. A LSTM node contains a memory cell and three gates that control how the cell and output should be influenced by the input data. A gate contains a neural network with a sigmoid function and a component wise operator. The input gate controls how much the input should affect the cell. The forget gate controls which information should be kept in the cell or removed. There is also an output gate that controls information in the output/hidden state. The gated cell and hidden state at time t are passed as input to the node that processes the input at time t+1 and so on. All the parameters that control the gates are defined through training.

In summary, LSTM has a selective memory that is trained to keep only the relevant information and this information is kept as long as it is required for the task. LSTMs can be stacked in order to improve its ability to process more complex classifications. For a detailed description of LSTM refer to (Hochreiter & Schmidhuber, 1997).

### 5 LSTM dependency parser

The LSTM parser follows the basic dependency parser structure. It supports the arc-standard and swap algorithms. It has a buffer B that is initialized with the input sentence. A stack S that contains partially built parse trees and a stack A of actions taken previously during the parsing process.

Figure 1: LSTM dependency parser with an example of kind of information stored in each component and the relationships between components. Adapted from (Dyer, Ballesteros, Ling, Matthews, & Smith, 2015).

In a standard LSTM, the output from the state at time t-1 \((h_{t-1}, c_{t-1})\) is used as input to calculate the state at time t. A stack LSTM has a pointer that defines which cell should be used to calculate the current state. The pointer sometimes is referred as the stack pointer. For example, the pointer could refer to the cell three steps behind \((h_{t-3}, c_{t-3})\) as input to calculate the state at time t.

An overview of LSTM dependency parser and the classifier function are presented in figure 1. Each data structure is represented by a corresponding stack LSTM that generates a representation for the data. The parser state at a certain time is defined by the contents of B, S, A. At each time step, the current parser state \((p_t)\) is used to define the next parser action that also implies changes in the parser state.

The state \(p_t\) at time t is defined by the expression based on a component-wise rectified linear unit (ReLU):

\[
p_t = max\{0, \quad W[s_t, b_t, a_t] + d\}
\]
The matrix W has its values defined during the training phase. The arrays $s_t, b_t, a_t$ are the LSTM representations for the $S, B, A$ data structures.

The S stack contains partial trees that must be defined in a vector representation before being processed by the LSTM. Recursive neural networks are used to create vector representations of the partial trees (Socher, et al., 2013).

**Internal word representation**

The representation of a word or token as a real-valued dense vector in the d-space ($\mathbb{R}^d$) is called word embedding.

The tokens in the tokenized input sentence are transformed into an internal embedded vector representation before they are processed by the LSTM parser. There are two approaches to process the input tokens. One approach is word based as described in (Dyer, Ballesteros, Ling, Matthews, & Smith, 2015). The other is character based as described in (Ballesteros, Dyer, & Smith, 2015). I am using the term word and token interchangeable unless there is a need to make a distinction.

5.1 **LSTM parser using word model**

This section is a brief summary of the LSTM parser based on word model (Dyer, Ballesteros, Ling, Matthews, & Smith, 2015). It has the same basic structure as defined in figure 1. However, when it processes the input words to create internal word representations, the principle is that each input word is treated as an indivisible entity. There are three vectors used as input to define the internal embedded word representation: a learned embedded representation for the input word ($w_{in}$), its POS tag ($pos_{in}$), and a fixed embedded representation for the input word ($w_{fix}$). The fixed embedded word representation is provided as input for training (e.g. word2vector generated). It is kept fixed during the whole training.

The internal embedded word representation ($x$) is defined by the following steps and equation: 1) all three vectors are concatenated, 2) they are passed through a linear transformation and 3) a component-wise ReLU generates the embedded representation.

$$x = max\{0, V[w_{in}, pos_{in}, w_{fix}] + d\}$$

The parameters used in this process are trained during the training phase.

The fixed embedded word representation is an optional parameter. If it is not present, only the other two vectors are used as input to calculate $x$.

A “UNK” word is used to represent out of vocabulary words (OOV) during training and parsing.

**Predefined embedded word representations**

Predefined embedded word representations are used as input for training. They are created using large amount of untagged text. There are different ways to create word embedding. One method is called skip-gram (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). A modified version of skip-gram model that preserves the word-order in a better way is described in (Ling, Dyer, Black, & Trancoso, 2015). It is called structured skip-gram and it has shown good results when handling syntax based problems.

5.2 **LSTM parser using character model**

This section is a brief summary of the LSTM parser based on character model (Ballesteros, Dyer, & Smith, 2015). It has the same basic structure as defined in figure 1. However, when it processes the input words to create internal word representations, the principle is that each input word is processed character by character. Each character is assigned a corresponding real valued vector representation that is called embedded character representation. A word embedding is defined based on the constituent characters embedding.

Figure 2 shows an example about how an input word is broken down into characters that are fed into two LSTMs. One LSTM processes character by character from left to right and the other LSTM processes character by character from right to left. The output from the two LSTMs and the corresponding POS tag are combined using a component-wise ReLU to produce the final internal embedded word representation (bottom of the picture). If the POS tag is not available (optional feature), only the other components are used.
Figure 2: Example of how the input word “rat” is processed character by character by two LSTMs to generate an embedded word representation. Adapted from (Ling, et al., 2015).

During the training phase, the embedded character representations are defined as well as all the parameters required to combine the characters into a corresponding word embedding. They are part of the model generated during training. When using the model to parse new sentences, the embedded character representation and the parameters are used in the same way to generate the embedded words.

Out of vocabulary (OOV) words are generated in as any other word using the embedded characters. That is, any kind of OOV; both correct as well as misspelled words can be composed. The generated OOV words are placed near by related words in the embedded d-space, according to (Ling, et al., 2015).

6 Experiments

6.1 Data

Universal Dependency Corpus

The experiments are based on the Universal Dependency corpus version 1.1 (Universal Dependencies 1.1, 2015). The main languages used were Swedish, English and Finnish. English and Finnish have approximately 12K sentences and Swedish has 4K.

Some experiments are based on a concatenated multi-language training and development corpus with 8 UD languages. The languages included were Bulgarian, Danish, German, English, Spanish, Persian, Finnish and Swedish. The concatenated training data has approximately 85K sentences.

Some experiments are based on English text of different lengths. The English UD corpus was split into five corpora of different sizes. The corpora have 2400 sentences, 4800 sentences, 7200 sentences, 9600 sentences and the whole corpus with 12K sentences.

Predefined Embedded words

The predefined embedded words for English created by (Dyer, Ballesteros, Ling, Matthews, & Smith, 2015) were used. The data is based on Agence France-Presse, English Service (AFP) subset of the English Gigaword corpus (version 5).

The latest Swedish Wikipedia dump available in 02 November 2015 was used to generate a Swedish embedded word representation. The gensim software python API (Řehůřek & Sojka, 2010) was the basis for the software to generate a Swedish embedded word representation. The Wikipedia text was extracted and tokenized. The text was normalized with only lower case and punctuation removed. The format used was skip-gram. The window size is equal to 5 and the dimension of the embedded word is equal to 100.

6.2 Configurations

The experiments used a LSTM parser based on word model and a LSTM parser based on character model. The following experiments were performed:

LSTM parser based on word model

The LSTM parser using a word based model was used with the standard configuration as defined in (Transition-based dependency parser based on stack LSTMs, 2015). Each LSTM cell has a dimension of 100 and each LSTM has two layers. The features used are the coarse POS tag and the input words. In some experiments predefined embedded word representations were used as input. The following experiments were executed:
1. English without word embedding
2. English with predefined word embedding
3. Swedish without word embedding
4. Swedish with predefined word embedding
5. Finnish without predefined word embedding

**LSTM parser based on character model**

The LSTM parser using character based model was used with the standard configuration (lstm-parser with character-based representations, 2015). Each LSTM cell has a dimension of 100. Each LSTM has two layers. The features used are the coarse POS tag and the input words. No pre-trained embedded word representation was used. The following experiments were performed:

6. English
7. English with multi-language model
8. Swedish
9. Swedish with multi-language model
10. Finnish
11. Finnish with multi-language model
12. English with different corpus size

**Evaluation principles**

In each case, the models were trained using the training and development sets available in UD corpus or the multi-language data.

Each model was always verified in the corresponding test set.

The scores are based on the eval.pl script and they do not include the punctuation. Both UAS and LAS scores are generated.

**6.3 Results and Analysis**

**LSTM parser based on word models without predefined word embedding generated:**

<table>
<thead>
<tr>
<th>language</th>
<th>UAS%</th>
<th>LAS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>88.44</td>
<td>85.87</td>
</tr>
<tr>
<td>Swedish</td>
<td>83.98</td>
<td>80.42</td>
</tr>
<tr>
<td>Finnish</td>
<td>73.37</td>
<td>66.80</td>
</tr>
</tbody>
</table>

Table 1: Results for parsing using word based LSTM and no predefined word embedding

The results are based on the corresponding UD test sets for each language. Table 1 shows that English has the best scores and Finnish the worse. This could be explained with the complexity of the languages. English has the simplest syntactic/morphological structure, Swedish has an increased syntax complexity (e.g. variable phrasal structure) and Finnish has the most complex morphological structure. Another factor that may have impacted Swedish results in a negative way is the small size of the training data. The size of the training data will be further analyzed.

**LSTM parser based on word models with predefined word embedding for Swedish and English:**

<table>
<thead>
<tr>
<th>language</th>
<th>UAS%</th>
<th>LAS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>88.35</td>
<td>85.81</td>
</tr>
<tr>
<td>Swedish</td>
<td>84.33</td>
<td>80.57</td>
</tr>
<tr>
<td>Finnish</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Results for parsing using word based LSTM and with predefined word embedding for Swedish and English

<table>
<thead>
<tr>
<th>language</th>
<th>UAS Δ%</th>
<th>LAS Δ%</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>-0.09</td>
<td>-0.06</td>
</tr>
<tr>
<td>Swedish</td>
<td>+0.35</td>
<td>+0.15</td>
</tr>
<tr>
<td>Finnish</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Difference (Δ) in percent between parsing with predefined word embedding (table2) and without (table 1)

The results are based on the corresponding UD test sets for each language. Tables 2 and 3 show that predefined word embedding does not provide any improvements in English and very modest improvements in Swedish when compared with the results in table 1. The Finnish data was not included in this measurement.

In general, the predefined word embedding should produce better results (Ling, Dyer, Black, & Trancoso, 2015). However, my results do not support this. One possible explanation is that the predefined word embedding used for English are based only on news articles (AFP) and they do not match very well the contents of the English UD corpus.
The word embedding for Swedish are based on Wikipedia that contains a more diverse text, therefore it may provide some modest improvements.

A negative factor may be the fact that I am using the word embedding based on standard skip-gram model. It is not the best model for syntactic analysis according to (Ling, Dyer, Black, & Trancoso, 2015).

Another negative factor may be the use of a window size of 5 when running the word2vec software. A smaller window size may produce a word embedding where syntactically similar words are grouped together tighter (Guo, Che, Yarowsky, Wang, & Liu, 2015).

**LSTM parser based on character models using own language training data:**

<table>
<thead>
<tr>
<th>language</th>
<th>UAS %</th>
<th>LAS %</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>87.21</td>
<td>84.55</td>
</tr>
<tr>
<td>Swedish</td>
<td>83.59</td>
<td>79.66</td>
</tr>
<tr>
<td>Finnish</td>
<td>81.71</td>
<td>77.60</td>
</tr>
</tbody>
</table>

Table 4: Results for parsing using character based LSTM

<table>
<thead>
<tr>
<th>language</th>
<th>UAS Δ %</th>
<th>LAS Δ %</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>-1.23</td>
<td>-1.32</td>
</tr>
<tr>
<td>Swedish</td>
<td>-0.39</td>
<td>-0.76</td>
</tr>
<tr>
<td>Finnish</td>
<td>+8.34</td>
<td>+10.80</td>
</tr>
</tbody>
</table>

Table 5: Difference (Δ) in percent between parsing with character model (table 4) and parsing with word model without predefined word embedding (table 1)

The results are based on the corresponding UD test sets for each language. Tables 4 and 5 show that there is a small decrease in the parsing accuracy for English and Swedish when compared with the word based LSTM parsers. There is a large improvement for Finnish. The character based LSTM parser seems to be able to identify detailed morphological patterns that are supporting the parsing of Finnish. It may also be taking advantage of the large Finnish training set. English and Swedish have a relatively simple morphology therefore processing whole words may produce the best reduces. It should be noted that Ballesteros et al. (2015) also got poorer results for English using character LSTM when compared to the word based LSTM parser.

**LSTM parser based on character models using multi-language training data:**

<table>
<thead>
<tr>
<th>language</th>
<th>UAS%</th>
<th>LAS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>87.53</td>
<td>84.93</td>
</tr>
<tr>
<td>Swedish</td>
<td>85.84</td>
<td>81.98</td>
</tr>
<tr>
<td>Finnish</td>
<td>82.03</td>
<td>77.91</td>
</tr>
</tbody>
</table>

Table 6: Results for parsing using character based LSTM trained with multi-language corpus

<table>
<thead>
<tr>
<th>language</th>
<th>UAS Δ %</th>
<th>UAS Δ %</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>+0.32</td>
<td>+0.38</td>
</tr>
<tr>
<td>Swedish</td>
<td>+2.25</td>
<td>+2.32</td>
</tr>
<tr>
<td>Finnish</td>
<td>+0.32</td>
<td>+0.31</td>
</tr>
</tbody>
</table>

Table 7: Difference (Δ) in percent between parsing with multi-language training corpus (table 6) and with specific language corpus (table 5). Both are cases are using character based parsers.

The multi-language model with 8 languages was used as training data. This was feasible because the character based LSTM parser model has a very compact memory footprint.

The results are based on the corresponding UD test sets for each language. Tables 6 and 7 show that scores for all languages get an improvement when compared to the character based LSTM scores. Swedish gets the best improvements and its best results.

One basic explanation is an increased search space for the LSTM with a very heterogeneous environment and the assumption that the dependencies have similarities among the languages. UD has a set of standard POS tags. The syntactic similarities (POS dependencies) among languages have been observed before (McDonald, Petrov, & Hall, 2011). The lexical similarities are much less clear. The common model builds a distributed character embedding that is multi-lingual. That is, each character and word contains contributions from many languages. It seems that the LSTM is able to extract relevant feature representations from this common representation. However, it is difficult to get an understanding of the patterns (syntactic and/or lexical) that are being extracted by the complex LSTMs.
Swedish is the language with the smallest training data and it gets the biggest improvement in parsing accuracy. That is, this multi-language model compensates for the shortage of training data in Swedish.

The drawback here is that the training data is large and the training is very time consuming. Therefore, this experiment was limited.

In this experiment, I have mixed languages with different alphabet (Persian, Bulgarian, English, etc.). However, the characters from the different alphabets will never get mixed. Therefore, a more practical solution should be to concatenate languages per kind of alphabet.

**LSTM parser based on character models using English corpora of different sizes:**

<table>
<thead>
<tr>
<th>Number of English sentences in training data</th>
<th>UAS%</th>
<th>LAS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2400</td>
<td>83.06</td>
<td>79.48</td>
</tr>
<tr>
<td>4800</td>
<td>83.82</td>
<td>80.30</td>
</tr>
<tr>
<td>7200</td>
<td>85.89</td>
<td>82.62</td>
</tr>
<tr>
<td>9600</td>
<td>86.68</td>
<td>83.81</td>
</tr>
<tr>
<td>12K (complete corpus)</td>
<td>87.21</td>
<td>84.55</td>
</tr>
</tbody>
</table>

Table 8: Results for parsing using character based LSTM and English corpora of different sizes

The number of sentences in the training data starts with 2400 sentences. It is increased in steps of 2400 sentences until all the training data is covered. One parser model for each corpus size was generated using the LSTM parser based on character model. The results for each model/corpus size are based on the complete English UD test set.

Table 8 and figure 3 show a continuous improvement of the scores with increased number of sentences for English. In figure 3 I show the scores for the Swedish text in the vertical line at 4000 sentences marked as SW. It is comparable with the scores for English text with 4800 sentences.

Further study is needed to find out how the size of the training data affects the LSTM score accuracy for other languages. Another interesting question is to find the optimal size for the training data. That is, find the smallest size of training data that produces best accuracy.

1 Filip Antomonov kindly provided the Stanford Neural Network results.

---

**Comparing LSTM Parsers and Stanford Neural Network for Swedish:**

The data available for the Stanford Neural Network parser is based on Swedish UD corpus version 1.2. The parser uses the pre-defined Swedish word embedding in SPMRL 2013 as input for training.

Table 9 presents the results for the Stanford parser and the results for the different LSTM parsers. All results are based on Swedish UD corpus. The LSTM parsers have better results than Stanford in all cases but the LAS for LSTM based on character.

<table>
<thead>
<tr>
<th>Swedish UD</th>
<th>UAS%</th>
<th>LAS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stanford NN with embedded word input</td>
<td>82.8</td>
<td>80.1</td>
</tr>
<tr>
<td>LSTM word model and NO embedded word input</td>
<td>83.98</td>
<td>80.42</td>
</tr>
<tr>
<td>LSTM word model with embedded word input</td>
<td><strong>84.33</strong></td>
<td><strong>80.57</strong></td>
</tr>
<tr>
<td>LSTM char model</td>
<td>83.59</td>
<td>79.66</td>
</tr>
</tbody>
</table>

Table 9: Comparison of Stanford and LSTM parsers using the Swedish UD for training and testing.
7 Related work

In general terms, it is well known that the complexity of the machine learning algorithm and the number of parameters have an impact on the size of training data. The more complex algorithms the more training data is needed, otherwise the classifier will over-fit (Abu-Mustafa, Magdon-Ismail, & Lin, 2012).

The study (Kirilin & Versley, 2015) has found that the size of the UD corpora affects the parsing accuracy when using Malt+Optimizer, MATE and RBG parsers.

The use of multi-language to support parsing of a new languages without any resources has been studied by (McDonald, Petrov, & Hall, 2011). They transfer delexicalized dependencies combined with aligned words in the different languages.

Another study attempts to transfer both lexical and syntactic feature with the support of word alignment and a number of features and a neural network classifier (Guo, Che, Yarowsky, Wang, & Liu, 2015).

8 Conclusion

I have shown that LSTM parsers with very simple features (coarse POS) have a reasonable performance on the UD corpora for English, Swedish and Finnish. The LSTM parser based on character model and the multi-language training data has the best results for Finnish and Swedish. See items a- and c- in the list below. The LSTM parser based on word model without predefined word embedding has the best performance for English. See item b- in the list below. Here is a summary of the best results:

- a- Swedish: UAS: 85.84% and LAS: 81.98%
- b- English: UAS: 88.44% and LAS: 85.87%
- c- Finnish: UAS: 82.03% and LAS: 77.91%

The size of the corpus is an issue affecting the parser performance. There is a clear evidence that the size of the UD English corpora affects the parsing accuracy when using LSTM based parsers. That is, the larger the training data, the better results. The impact of the size of the training data in LSTM scores for other languages needs further study.

I have shown that it is possible to use multi-language corpora as input for training in order to improve the parsing accuracy for the languages with short training corpora. One area that should be further studied is the feasibility to develop one model per type of alphabet.

When parsing Swedish UD corpus, the LSTM parsers have a better performance than the Stanford neural network parser in all cases but one. The only exception is the LAS score for the character based LSTM.

9 References


