Investigating the Impact of Syntactic Features for Semantic Dependency Parsing

Maximilian Pfundstein

January 10, 2020

The theoretical baseline model provided by Dozat and Manning (Zeman et al. 2017) and the implementation from Daniel Roxbo (Roxbo 2019) are used for predicting syntactic and semantic dependency graphs. For both, one BiLSTM is being trained and their performance is evaluated in a multi-task learning setup. The original model, semantic only, results in an F_1 score of 87.50% for an out-of-domain test data set. The multitask model using raw edge matrices achieves an F_1 score of 87.80% and another approach using a feedback loop, providing previous learnt semantic as an input for the syntactic scorer, results in an F_1 score of 87.87%. Thus, multitask learning improves the results, but not significantly. Additionally, the implementation from Daniel Roxbo (ibid.) has been updated to provide faster startup times and the ability to read `.conllu` files. An existing library called *CoNLL-U Parser* is being used for this functionality and some features have been provided for the library via pull-requests on GitHub.

Contents

Scope

The purpose of this report is to summarise the obtained results during the research project (732A76) which is a course of the masters programme in *Statistics and Machine Learning* at Linköpings University in 2019. It therefore assumes that the reader is familiar with the named references, the code base and terms associated with it.

1 Introduction

Syntactic and semantic dependency parsing (SDP) within the domain of NLP (Natural Language Processing) is the task of creating a directed acyclic graph consisting of nodes and edges to embed the grammatical meaning of a sentence. The research question being investigated here is, if for the prediction of semantic dependencies syntactic information can be utilised and to which extend they provide information for increasing the accuracy of a model.

For answering that question and conducting research, the baseline model originally provided by Dozat and Manning (Zeman et al. 2017) is being used. The replicated implementation from Daniel Roxbo (Roxbo 2019) has been extended during his thesis to be able to also learn syntactic graphs which are provided as features for the semantic dependency parser. The results show that while syntactic gold graphs increase the performance of the semantic parser from 93.6 percent to 97.1 percent, predicted syntactic trees yield in a slightly lower F_1 score of 92.8 percent (ibid., p. 33). These scores relate to in-domain data.

Further research by Kurtz, Roxbo, and Kuhlmann 2019 suggests that there is an overlap between syntactic and semantic information. This would explain the results provided by (Roxbo 2019, p. 33) and indicate that no further information can be extracted. Other research suggests, that multitask learning can actually improve the results of a solely semantic parser (Peng, Thomson, and Smith 2017). There, the same theoretical model is being used which was applied by Dozat and Manning (Zeman et al. 2017) and the data utilised is the same as used by Roxbo 2019. Then, multitask learning is being applied and higher F_1 scores, compared to a solely semantic parser, are achieved.

It should be noticed that the results from Peng, Thomson, and Smith 2017 are lower for the multitask learning compared to the pure semantic parser implemented by Robin Roxbo (Roxbo 2019). This might be due to a smaller model or a difference in implementation. This research project will not dive deeper into the differences of the implementations, but will focus on examining a multitask learning approach using the implementation from Robin Roxbo (ibid.) to investigate, if the semantic parser can be further improved using predicted syntax.

To further elaborate on this, multitask learning will be utilised to learn the syntactic and semantic scores at the same time while using an interpolated loss. Additionally, another extension to this approach will be tested, which will feed the semantic output back to the syntactic scorer.

2 Overview

This research project[1] is embedded in a course of the Masters Programme in Statistics and Machine Learning at Linköpings University[2] in 2019 and represents a course[3].

[1] https://www.ida.liu.se/ 732A76/index.en.shtml
[2] https://liu.se/en/education/program/f7msl
[3] https://www.ida.liu.se/~732A76/info/courseinfo.en.shtml

Therefore, this report also contains improvements to the existing code base which will also be presented in the following chapters. This mainly includes improvements to the existing code base.

The official objectives for this research project have been formalised in the following way:

> The goal of the project is to investigate the impact of syntactic features for semantic dependency parsing. While it is already known that syntactic features help, previous experiments have shown that predicting syntactic structure, and to then use it as a feature for semantic parsing does not increase performance.
>
> The role of the student is to modernize an existing system, adjust it to read combined syntax-semantic input data, and investigate how to successfully add syntactic features to semantic dependency parsing. This involves testing different neural network variations, pipeline processing and multitask learning.

During the project the following three objectives have been formulated:

- Allow to import the data from the `.conllu` format.
- Increase the performance during the pre-processing by using caching to save computation time.
- Implement multitask learning to simultaneously learn syntactic and semantic graphs and investigate the results.

The following chapter will explain the different tasks further and which work has been done in detail. The subsequent chapter will present the results of the three objectives and the final chapter includes discussions, conclusions and suggestions.

3 Objectives

This chapter will introduce which work has be done for all of the objectives of this research project.

3.1 .conllu Format

The implementation from Robin Roxbo (Roxbo 2019) reads the syntactic graphs from `.cpn` and the semantic graphs respectively from `.sdp` files. Both formats can be seen in figure 1 and figure 2. The aim is to support the CoNLL-U format[4] which is based on Buchholz and Marsi 2006. The format embeds both syntactic and semantic graphs and related information. Furthermore, the support for `.cpn` and `.sdp` is being discontinued.

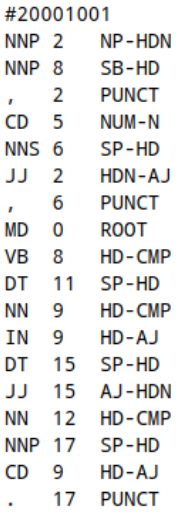

Figure 1: `.cpn` format (syntax)

The data is provided in `.conllu` format along with a sparse implementation for parsing. To fully support `.conllu` it has to be decided if the already present code is being further utilised, rewritten or if other implementations exist. The library *CoNLL-U Parser*[5] is freely available on GitHub, supports all necessary features and has a good test coverage. Therefore it has been decided to adopt the library and use it for the existing code base.

During the implementation two problems have been encountered. First, the labels for the edges contain uppercase letters and numbers (compare figure 3) which are not officially supported by the CoNLL-U specification. After discussion with the maintainer of the library it was decided to add support for uppercase letters and numbers as there was, apart from not strictly following the specification, no argument against it. The changes[6] have been submitted via pull-request[7] on GitHub.

Secondly, the sentence Ids, which precede each sentence in all formats, are also contrary to the specification. The Ids can be seen in figure 1, figure 2 and figure 3 in the first row respectively. After evaluation with the library maintainer, a custom fallback parser has been added by the maintainer to handle arbitrary formats of meta information[8]. Furthermore, the row containing the ill-formatted sentence Id has been refactored to be in accordance with the specification. Thus, the row containing the valid sentence Id has the following format after the change: `# sent-id = 20001001`. A small Python script has been written to reformat the existing `.conllu` files. It can be found in ap-

[4]https://universaldependencies.org/format.html
[5]https://github.com/EmilStenstrom/conllu
[6]Issue: https://github.com/EmilStenstrom/conllu/issues/34
[7]Pull Request: https://github.com/EmilStenstrom/conllu/commit/532624b1b897d6167e0da99b4944c345ae04609c
[8]https://github.com/EmilStenstrom/conllu#customizing-parsing-to-handle-strange-variations-of-conll-u

```
#SDP 2015
#20001001
1   Pierre  Pierre  NNP -   +   named:x-c   _   _   _   _   _   _   _   _   _   _   _
2   Vinken  _generic_proper_ne_ NNP -   -   named:x-c   compound    _   _   ARG1    ARG1    _   _   _   _   _   _
3   ,   _   ,   -   -   _   _   _   _   _   _   _   _   _   _   _
4   61  _generic_card_ne_   CD  -   +   card:i-i-c  _   _   _   _   _   _   _   _   _   _   _
5   years   year    NNS -   +   n:x _   ARG1    _   _   _   _   _   _   _   _
6   old old JJ  -   +   a:e-p   _   _   measure _   _   _   _   _   _   _   _
7   ,   _   ,   -   -   _   _   _   _   _   _   _   _   _   _   _
8   will    will    MD  -   -   _   _   _   _   _   _   _   _   _   _   _
9   join    join    VB  +   +   v:e-i-p _   _   _   _   _   ARG1    _   _   _   loc
10  the the DT  -   +   q:i-h-h _   _   _   _   _   _   _   _   _   _
11  board   board   NN  -   -   n_of:x-i    _   _   _   _   ARG2    BV  _   _   _   _   _
12  as  as  IN  -   +   p:e-u-i _   _   _   _   _   _   _   _   _   _   _
13  a   a   DT  -   +   q:i-h-h _   _   _   _   _   _   _   _   _   _   _
14  nonexecutive    _generic_jj_    JJ  -   +   a:e-u   _   _   _   _   _   _   _   _   _   _   _
15  director    director    NN  -   -   n_of:x-i    _   _   _   _   _   _   ARG2    BV  ARG1    _   _
16  Nov.    Nov.    NNP -   +   mofy:x-c   _   _   _   _   _   _   _   _   _   _   _
17  29  _generic_dom_card_ne_   CD  -   +   dofm:x-c   _   _   _   _   _   _   _   _   _   of  _
18  .   _   .   -   -   _   _   _   _   _   _   _   _   _   _   _
```

Figure 2: `.sdp` format (semantic)

pendix 6.1.

3.2 Performance Increase

Every time the source code is being invoked, the pre-processing is performed again which causes a delay of a few minutes before the task can be executed. Here a task can be for example training or predicting. As the steps for the pre-processing are the same every time, the results are supposed to be cached.

An investigation of the source code led to three functions which are being modified to support caching. These are `parse_conllu_labels`, `parse_conllu_sentences` and `parse_conllu_targets`. Pythons `pickle` [9] module is being used to cache the results to disk, where the binary format is then being loaded when the method is being invoked again. Some methods are being invoked multiple times, for example when called for the training and development data sets. In addition, an option has been added to disable caching if desired.

3.3 Multitask Learning

As suggested in Kurtz, Roxbo, and Kuhlmann 2019, multitask learning is implemented for evaluating if predicted syntax can improve the semantic score in this setting. Two approaches have been implemented and will be presented in this chapter. As the model is likely to capture more information, the hyperparameters have been adjusted to account for that. In particular, the 3 layers of the LSTM (Long Short-Term Memory) have been increased extended to 4 layers and the `hidden_lstm` and `hidden_char_lstm` have been increased from 400 to 600. The full list of hyperparameters can be found in appendix 6.2.

[9] https://docs.python.org/3/library/pickle.html

```
#22000001
1   Rockwell    _generic_proper_ne_ _   NNP _   2   NP-HDN  _   _
2   International   _generic_proper_ne_ _   NNP _   4   SP-HD   1:compound|3:compound   _
3   Corp.   corp.   _   NNP _   2   HDN-AJ  _   _
4   's  's  _   VBZ _   6   SP-HD   _   _
5   Tulsa   _generic_proper_ne_ _   NNP _   6   NP-HDN  _   _
6   unit    unit    _   NN  _   7   SB-HD   2:poss|5:compound|7:ARG1    _
7   said    say _   VBD _   0   ROOT    0:ROOT  _
8   it  it  _   PRP _   9   SB-HD   9:ARG1  _
9   signed  sign    _   VBD _   7   HD-CMP  13:subord   _
10  a   a   _   DT  _   12  SP-HD   _   _
11  tentative   tentative   _   JJ  _   12  AJ-HDN  _   _
12  agreement   agreement   _   NN  _   9   HD-CMP  9:ARG2|10:BV|11:ARG1    _
13  extending   extend  _   VBG _   9   HD-AJ   7:ARG2  _
14  its its _   PRP$    _   15  SP-HD   _   _
15  contract    contract    _   NN  _   13  HD-CMP  13:ARG2|14:poss|16:ARG1 _
16  with    with    _   IN  _   15  HDN-AJ  _   _
17  Boeing  Boeing  _   NNP _   16  HD-CMP  16:ARG2|18:compound _
18  Co. co. _   NNP _   17  HDN-AJ  _   _
19  to  to  _   TO  _   15  HDN-AJ  _   _
20  provide provide _   VB  _   19  HD-CMP  _   _
21  structural  structural  _   JJ  _   22  AJ-HDN  _   _
22  parts   part    _   NNS _   20  HD-CMP  20:ARG2|21:ARG1|23:ARG1 _
23  for for _   IN  _   22  HDN-AJ  _   _
24  Boeing  Boeing  _   NNP _   25  SP-HD   _   _
25  's  's  _   VBZ _   27  SP-HD   _   _
26  747 _generic_card_ne_   _   CD  _   27  AJ-HDN  _   _
27  jetliners   _generic_nns_   _   NNS _   23  HD-CMP  23:ARG2|24:poss|26:ARG1 _
28  .   _   _   .   _   27  PUNCT   _   _
```

Figure 3: `.conllu` format (semantic and syntactic)

3.3.1 Interpolated Score

The same model is being used twice, once for predicting syntax and once for predicting semantic. First, the model for predicting syntax is being called and the resulting predictions are then utilised as input features for the semantic predictions. The process structure can be seen in figure 4. Then, a loss is being calculated for each model and the resulting loss is an interpolation between both, which can then be set as a hyperparameter. Both the syntactic and semantic scorer use the same internal representation of information which is the output of the BiLSTM .

Three different variations have been tested. *Multitask Raw* passes the raw syntax edge scores forward, which are positive and negative floating point numbers. *Multitask Single* sets only for each row of the score matrix (thus for each word) the highest edge score to 1.0, whereas *Multitask Multi* sets all edges to 1.0 which are greater than 1.0. The idea behind this approach is that the semantic scorer potentially will benefit from a simpler input representation.

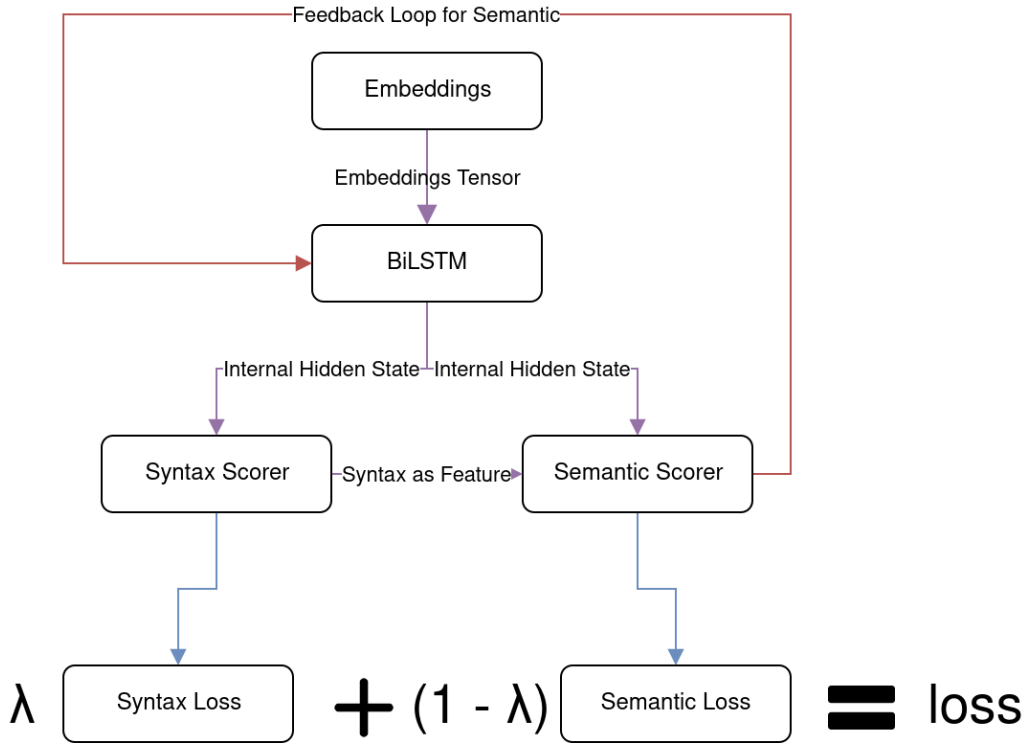

Figure 4: Flowchart, with red representing the feedback loop.

3.3.2 Feedback Loop

Building on top of the same concept as the interpolated score, the feedback loop approach performs one forward pass first, for each batch, and then feeds the semantic predictions as features to the syntactic input. This way the syntax scorer might be able to use learned semantic information from previous pass-throughs for further improvement. The structure can be seen in figure 4, where the red line represents the feedback loop for the semantic information passed back to the BiLSTM so that the syntactic scorer can use it.

The feedback loop was only tested for *Multitask Raw* as time was limited and *Multitask Raw* looked most promising during testing.

4 Results

The results of the three objectives will be presented in this chapter.

4.1 `.conllu` Format

All discussed changes to the library have been adopted by the maintainer of the library. The code base has been modified at several locations, especially as many functions had dependencies on the previous existing tooling functions for reading in the input. Some of the dependencies have been resolved and the code base can now fully rely on the existing library for reading `.conllu` files.

4.2 Performance Improvement

The startup time for the code base could be improved by several seconds. Depending on the task to perform, sometimes even by minutes. Figure 5 shows the speedup achieved after implementation of caching for the three main functions of the code base.

The function `parse_conllu_labels` benefited most from the caching, reducing the computational time to 341μs. For the other functions the speedup was not as significant, but still saves several seconds each run. `parse_conllu_sentences` is around 7.93 times faster and the function `parse_conllu_targets` is around 9.42 times faster than before.

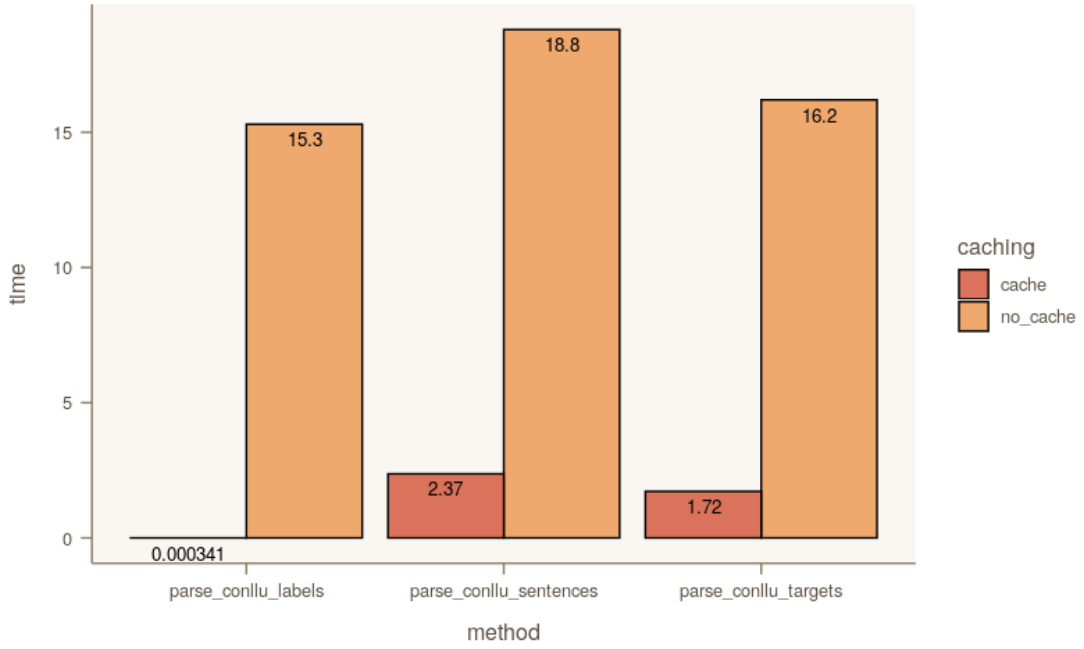

Figure 5: Speedup using caching (time in seconds).

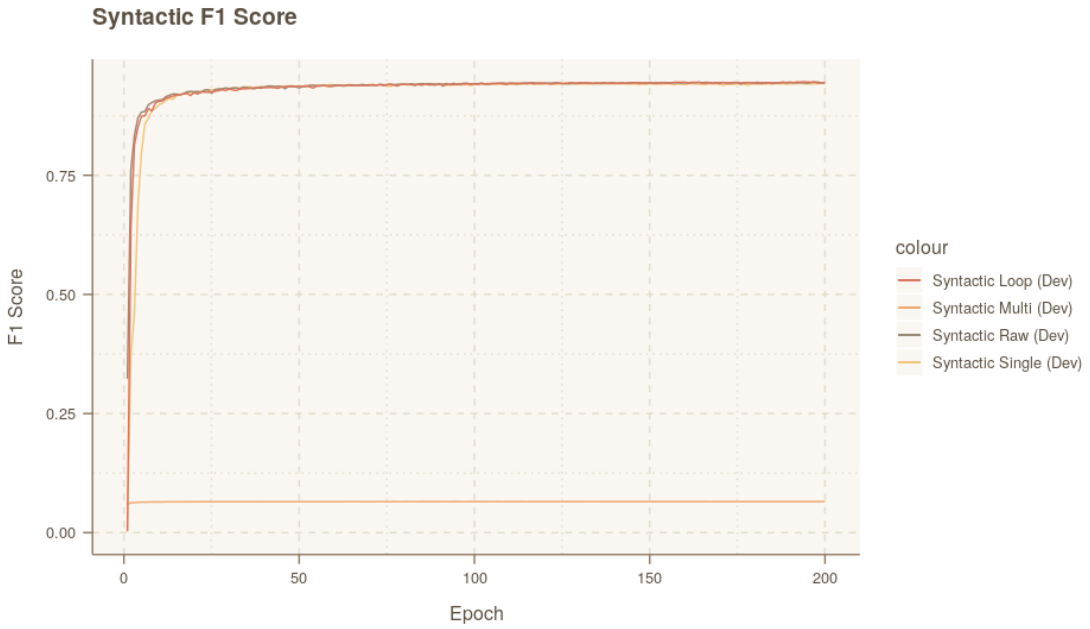

Figure 6: Syntactic F_1 score for 200 epochs.

4.3 Multitask Learning

The F_1 scores on the development data set during training can be seen in figure 6 for the syntactic F_1 scores and in figure 8 for the semantic F_1 scores over 200 epochs. Figure 7 and 9 show the curves for the last 50 epochs.

It can be seen that for the syntactic score, using multiple edges set to 1.0 (Syntactic Multi), the networks completely fails to learn. For setting just one edge to 1.0, the network is learning, but it can be seen that it learns slower and reaches a lower F_1 score in the end compared to both implementations using raw edge matrices. We therefore conclude that reducing any information about scores yields in lower F_1 scores and the assumption, that the semantic scorer benefits from a simpler representation, is wrong.

All semantic versions of the network manage to learn to some good extend. The single semantic version learns slowest while also performing worst. These observations are consistent with the results for the syntactic scorer.

Looking at the last 50 epochs for the semantic scores, it can actually be seen that the different scorers stagnate at different values. A conducted one way ANOVA test also shows with very high significance ($p < 10^{-6}$), that all scorers do not share a common mean for the F_1 score. As for the syntactic scorers, the raw edge matrices perform best. Interestingly, the feedback loop seems have a slightly higher F_1 score compared to the raw multitask version. Nevertheless, the improvement is very marginal.

Table 1 shows the final F_1 scores for syntax and semantic models for the development and test data set from the best models during the training process. Missing values are

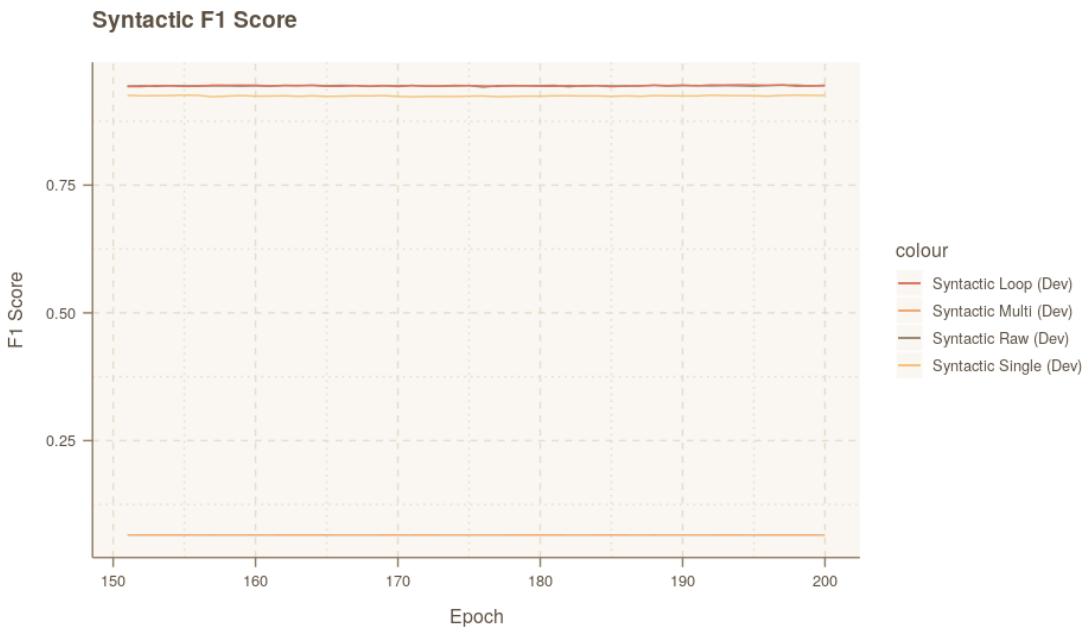

Figure 7: Syntactic F_1 score for the last 50 epochs.

denoted with a *minus* and undefined values are defined by the letter *o*. The missing values are available on request, as all models have been saved, but the values have not been logged during evaluation.

It can be seen that multitask learning seems to slightly increase the semantic test score. The *Semantic Only* model achieves an F_1 score of 87.50% whereas the multitask learning model using raw edges achieves an F_1 score of 87.80% and the feedback loop model achieves a very slightly higher F_1 score of 87.87%.

5 Conclusions

It seems that multitask learning does increase the performance, but not as much as proposed by Peng, Thomson, and Smith 2017. At least in terms of this data set. One of the reasons for this behaviour could be that the model used by ibid. underfits the data, as the presented semantic only scorer (*Semantic Only*) already outperforms the proposed multitask model from ibid. It could be that the information captured using a multitask approach can also be learned by a model not utilising multitask learning.

The findings of this research project emphasise the presumption of Kurtz, Roxbo, and Kuhlmann 2019 that the syntactic and semantic representation of a sentence indeed share some information.

To further investigate this presumption, one could take a look at Bidirectional Encoder Representations from Transformers (BERT) and see if this pre-trained model

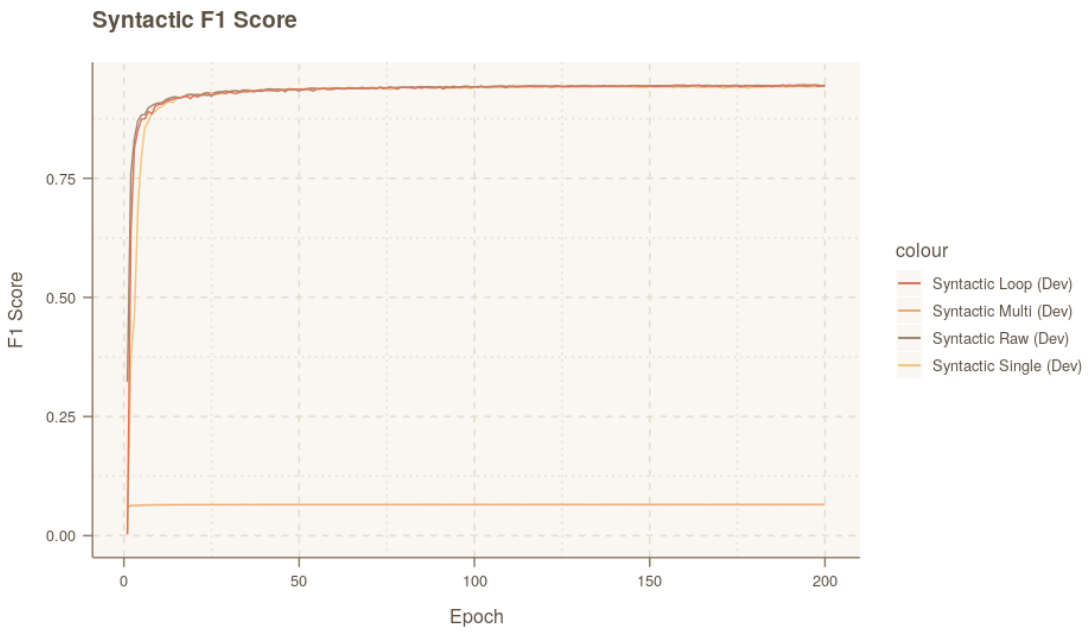

Figure 8: Semantic F_1 score for 200 epochs.

achieves higher scores compared to using Global Vectors for Word Representation (GloVe) embeddings. If computational resources are available, BERT could be trained on this specific domain of problem. Another approach could be to combine BERT with multi-task learning.

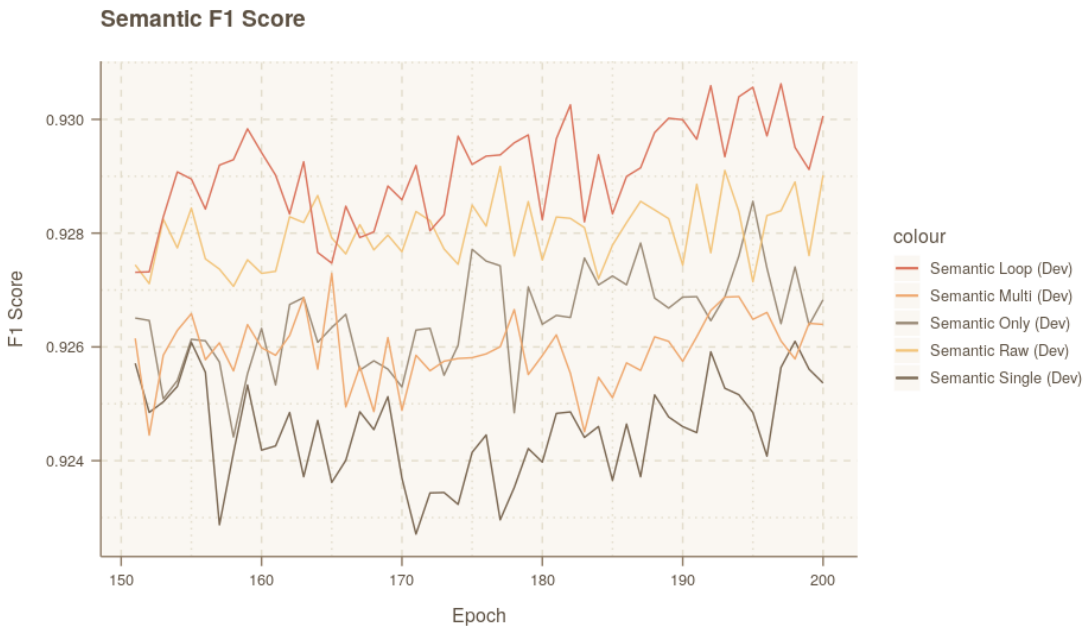

Figure 9: Semantic F_1 score for the last 50 epochs.

	In-Domain				Out-Of-Domain	
Setting	**Syn Dev**	**Sem Dev**	**Syn Test**	**Sem Test**	**Syn Test**	**Sem Test**
Sem Only	0	92.68%	0	92.39%	0	**87.50%**
Multitask Raw	94.54%	92.90%	-	-	90.92%	**87.80%**
Multitask Single	94.27%	92.54%	94.64%	92.60%	90.81%	87.41%
Multitask Multi	6.52%	92.64%	-	-	7.39%	87.71%
Multitask Loop	94.42%	93.01%	94.29%	92.51%	90.96%	**87.87%**

Table 1: F_1 scores for the different setups.

References

Buchholz, Sabine and Erwin Marsi (2006). "CoNLL-X Shared Task on Multilingual Dependency Parsing". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning*. CoNLL-X '06. New York City, New York: Association for Computational Linguistics, pp. 149–164.

Peng, Hao, Sam Thomson, and Noah A. Smith (July 2017). "Deep Multitask Learning for Semantic Dependency Parsing". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 2037–2048. DOI: 10.18653/v1/P17-1186. URL: https://www.aclweb.org/anthology/P17-1186.

Zeman, Daniel et al. (Aug. 2017). "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*. Vancouver, Canada: Association for Computational Linguistics, pp. 1–19. DOI: 10.18653/v1/K17-3001. URL: https://www.aclweb.org/anthology/K17-3001.

Kurtz, Robin, Daniel Roxbo, and Marco Kuhlmann (Sept. 2019). "Improving Semantic Dependency Parsing with Syntactic Features". In: *Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing*. Turku, Finland: Linköping University Electronic Press, pp. 12–21. URL: https://www.aclweb.org/anthology/W19-6202.

Roxbo, Daniel (2019). "A Detailed Analysis of Semantic Dependency Parsing with Deep Neural Networks". MA thesis. Linköping University, Human-Centered systems, p. 47.

6 Appendix

6.1 Python Script for Reformatting Sentence Id Rows

```python
import sys
import os

read_path = sys.argv[1]
write_path = sys.argv[1] + ".temp"
with open(read_path, "r", encoding="utf-8") as read_file:
    with open(write_path, "w", encoding="utf-8") as write_file:
        for line in read_file:
            if line[0] == "#" and line[1] != " ":
                line = line[0] + " sent-id = " + line[1:]
            write_file.write(line)

os.rename(write_path, read_path)
```

6.2 Hyperparameters

6.2.1 Semantic Only

```
[data]
train                   = data/sdp/en.train.dm.dt.deepbank.conllup
val                     = data/sdp/en.dev.dm.dt.deepbank.conllup
predict_file            = data/sdp/en.id.dm.dt.deepbank.conllup
glove                   = data/glove.6B.100d.txt
#load                    =
syn_input_style         = none

[training]
batch_size              = 50
epochs                  = 200
beta1                   = 0
beta2                   = 0.95
l2                      = 3e-09

[network_sizes]
hidden_lstm             = 600
hidden_char_lstm        = 600
layers_lstm             = 4
dim_mlp                 = 600
dim_embedding           = 100
dim_char_embedding      = 100
early_stopping          = 0

[network]
```

```
pos_style               = xpos
target_style            = sem
attention               = bilinear
synsem_interpolation    = 0.0
loss_interpolation      = 0.025
lstm_implementation     = drop_connect
char_implementation     = convolved
disable_gradient_clip   = False
unfactorized            = True
emb_dropout_type        = replace
score_encoding          = raw

[features]
disable_glove           = False
disable_char            = False
disable_lemma           = True
disable_pos             = False
disable_form            = False

[dropout]
dropout_embedding       = 0.2
dropout_edge            = 0.25
dropout_label           = 0.33
dropout_main_recurrent  = 0.25
dropout_recurrent_char  = 0.33
dropout_main_ff         = 0.45
dropout_char_ff         = 0.33
dropout_char_linear     = 0.33

[other]
seed                    = 1234
force_cpu               = False

[output]
quiet                   = False
save_every              = False
disable_val_eval        = False
enable_train_eval       = False
```

6.2.2 Multitask Single

```
[data]
train                   = data/sdp/en.train.dm.dt.deepbank.conllup
val                     = data/sdp/en.dev.dm.dt.deepbank.conllup
predict_file            = data/sdp/en.id.dm.dt.deepbank.conllup
glove                   = data/glove.6B.100d.txt
#load                    =
```

```
syn_input_style         = gold

[training]
batch_size              = 50
epochs                  = 200
beta1                   = 0
beta2                   = 0.95
l2                      = 3e-09

[network_sizes]
hidden_lstm             = 600
hidden_char_lstm        = 600
layers_lstm             = 4
dim_mlp                 = 600
dim_embedding           = 100
dim_char_embedding      = 100
early_stopping          = 0

[network]
pos_style               = xpos
target_style            = syn+sem
attention               = bilinear
synsem_interpolation    = 0.4
loss_interpolation      = 0.025
lstm_implementation     = drop_connect
char_implementation     = convolved
disable_gradient_clip   = False
unfactorized            = True
emb_dropout_type        = replace
score_encoding          = single

[features]
disable_glove           = False
disable_char            = False
disable_lemma           = True
disable_pos             = False
disable_form            = False

[dropout]
dropout_embedding       = 0.2
dropout_edge            = 0.25
dropout_label           = 0.33
dropout_main_recurrent  = 0.25
dropout_recurrent_char  = 0.33
dropout_main_ff         = 0.45
dropout_char_ff         = 0.33
dropout_char_linear     = 0.33

```

```
[other]
seed                    = 1234
force_cpu               = False

[output]
quiet                   = False
save_every              = False
disable_val_eval        = False
enable_train_eval       = False
```

6.2.3 Multitask Raw

```
[data]
train                   = data/sdp/en.train.dm.dt.deepbank.conllup
val                     = data/sdp/en.dev.dm.dt.deepbank.conllup
predict_file            = data/sdp/en.id.dm.dt.deepbank.conllup
glove                   = data/glove.6B.100d.txt
#load                    =
syn_input_style         = gold

[training]
batch_size              = 50
epochs                  = 200
beta1                   = 0
beta2                   = 0.95
l2                      = 3e-09

[network_sizes]
hidden_lstm             = 600
hidden_char_lstm        = 600
layers_lstm             = 4
dim_mlp                 = 600
dim_embedding           = 100
dim_char_embedding      = 100
early_stopping          = 0

[network]
pos_style               = xpos
target_style            = syn+sem
attention               = bilinear
synsem_interpolation    = 0.4
loss_interpolation      = 0.025
lstm_implementation     = drop_connect
char_implementation     = convolved
disable_gradient_clip   = False
unfactorized            = True
emb_dropout_type        = replace
```

```
score_encoding          = raw

[features]
disable_glove           = False
disable_char            = False
disable_lemma           = True
disable_pos             = False
disable_form            = False

[dropout]
dropout_embedding       = 0.2
dropout_edge            = 0.25
dropout_label           = 0.33
dropout_main_recurrent  = 0.25
dropout_recurrent_char  = 0.33
dropout_main_ff         = 0.45
dropout_char_ff         = 0.33
dropout_char_linear     = 0.33

[other]
seed                    = 1234
force_cpu               = False

[output]
quiet                   = False
save_every              = False
disable_val_eval        = False
enable_train_eval       = False
```

6.2.4 Multitask Multi

```
[data]
train                   = data/sdp/en.train.dm.dt.deepbank.conllup
val                     = data/sdp/en.dev.dm.dt.deepbank.conllup
predict_file            = data/sdp/en.id.dm.dt.deepbank.conllup
glove                   = data/glove.6B.100d.txt
#load                    =
syn_input_style         = gold

[training]
batch_size              = 50
epochs                  = 200
beta1                   = 0
beta2                   = 0.95
l2                      = 3e-09

[network_sizes]
```

```
hidden_lstm             = 600
hidden_char_lstm        = 600
layers_lstm             = 4
dim_mlp                 = 600
dim_embedding           = 100
dim_char_embedding      = 100
early_stopping          = 0

[network]
pos_style               = xpos
target_style            = syn+sem
attention               = bilinear
synsem_interpolation    = 0.4
loss_interpolation      = 0.025
lstm_implementation     = drop_connect
char_implementation     = convolved
disable_gradient_clip   = False
unfactorized            = True
emb_dropout_type        = replace
score_encoding          = multi

[features]
disable_glove           = False
disable_char            = False
disable_lemma           = True
disable_pos             = False
disable_form            = False

[dropout]
dropout_embedding       = 0.2
dropout_edge            = 0.25
dropout_label           = 0.33
dropout_main_recurrent  = 0.25
dropout_recurrent_char  = 0.33
dropout_main_ff         = 0.45
dropout_char_ff         = 0.33
dropout_char_linear     = 0.33

[other]
seed                    = 1234
force_cpu               = False

[output]
quiet                   = False
save_every              = False
disable_val_eval        = False
enable_train_eval       = False
```

6.2.5 Multitask Loop

```
[data]
train                   = data/sdp/en.train.dm.dt.deepbank.conllup
val                     = data/sdp/en.dev.dm.dt.deepbank.conllup
predict_file            = data/sdp/en.id.dm.dt.deepbank.conllup
glove                   = data/glove.6B.100d.txt
#load                    =
syn_input_style         = gold

[training]
batch_size              = 50
epochs                  = 200
beta1                   = 0
beta2                   = 0.95
l2                      = 3e-09

[network_sizes]
hidden_lstm             = 600
hidden_char_lstm        = 600
layers_lstm             = 4
dim_mlp                 = 600
dim_embedding           = 100
dim_char_embedding      = 100
early_stopping          = 0

[network]
pos_style               = xpos
target_style            = syn+sem
attention               = bilinear
synsem_interpolation    = 0.4
loss_interpolation      = 0.025
lstm_implementation     = drop_connect
char_implementation     = convolved
disable_gradient_clip   = False
unfactorized            = True
emb_dropout_type        = replace
score_encoding          = raw

[features]
disable_glove           = False
disable_char            = False
disable_lemma           = True
disable_pos             = False
disable_form            = False

[dropout]
dropout_embedding       = 0.2
```

```
dropout_edge           = 0.25
dropout_label          = 0.33
dropout_main_recurrent = 0.25
dropout_recurrent_char = 0.33
dropout_main_ff        = 0.45
dropout_char_ff        = 0.33
dropout_char_linear    = 0.33

[other]
seed                   = 1234
force_cpu              = False

[output]
quiet                  = False
save_every             = False
disable_val_eval       = False
enable_train_eval      = False
```