UPPSALA UNIVERSITET

Improving Implicit Sense Classification in Shallow Discourse Parsing

Jimmy Callin

Uppsala University
Department of Linguistics and Philology
Master's Programme in Language Technology
Master's Thesis in Language Technology

October 20, 2016

Supervisors:
Joakim Nivre

Abstract

TODO

Contents

Preface

1 Introduction

Despite the progress in Natural Language Processing (NLP), one assumption remains steadfast: sentences are to be seen as atomical units. Machine Translation systems rarely look outside of its local context for translation clues, sentiment analysis modules have trouble dealing with juxtapositions, and information extraction systems often limit extraction to the immediate sentence. While this may seem like an odd assumption to make, it has been difficult to release this constraint from the equations; the increasing recall has rarely been worth the much faster decreasing precision.

Lately, it has come to the attention that we might have started to reach a convergence for certain tasks if we do not start to look into the inherent structure between sentences within documents. That is, how does one text unit relate to the another? These *discourse relations*, as they often are called, is the main target for *discourse parsing*.

Take for instance:

(1) *Boeing would make cost-of-living adjustments projected to be 5 for each year of the contract* though **the union has called the offer insulting**.

The connective unit *though* in Example 5 can be analyzed as a binding block between the italized unit and the bolded, expressing a *contrastive comparison*. This is an example of a *sense* of a discourse relation which labels each relation according to a given sense taxonomy. That said, not all connective units are necessarily as explicit:

(2) *No wonder*. **We were coming down straight into their canal.**

Here, despite the lack of a connective unit the two sentences clearly share a connection. The bolded sentence explains the previous one as *the reason* for the lack of wonder. We will expand upon the difference between implicit and explicit relations in section 2.0.1, as well as how we can possibly classify the sense of a relation.

1.1 Purpose

The purpose of this thesis is to explore methods of improving sense classification in English Shallow Discourse Parsing. The main research question to explore is *how can we use continuous semantic representations to increase performance in such a task?* In particular, *what benefits can we get from applying neural network architectures with continuous semantic representations?*

2 Background

2.0.1 Shallow Discourse Parsing

[THIS IS HOW DISCOURSE PARSING HAS TRADITIONALLY BEEN DONE THROUGH THE WORKS OF RHETORICAL STRUCTURE AND SEGMENTED DISCOURSE REPRESENTATION THEORY].

[THIS IS WHAT SHALLOW DISCOURSE PARSING TRIES TO DO DIFFERENTLY].

The shallow property is realized by the lack of a hierarchical structure, and also by not requiring total coverage of all text units within a document. This stands in contrast to other discourse frameworks, with the purpose that [WHY?].

Penn Discourse Treebank

The Penn Discourse Treebank (PRASAD ET AL 2008) is a subset of the Penn Treebank annotated according to the framework as specified by WEBBER (2004). It follows a lexically-grounded approach to annotation of discourse relations (WEBBER ET AL 2003), and adopts a theory-neutral approach to annotation. It makes no assumptions on higher level structures beyond the lexically constrained annotation and their sense relation.

The PDTB framework defines two arguments, *Arg1* and *Arg2*, which make up the two text units that are connected by a discourse relation. These two arguments may or may not be connected by a *discourse connective*. If a discourse connective is present we call it an *explicit* discourse relation – otherwise the relation is *implicit*. A discourse connective is generally one or more words that binds the arguments together, e.g. by a subordinate conjunction such as *because*, *and*, or *however*. If an explicit connective is present, whichever argument it is syntactically connected to is defined as *Arg1*, while the other is *Arg2*.

Discourse connectives

See Table 2.1 for discourse type distribution. See Table 2.2 for sense hierarchy.

Explicit vs implicit discourse connectives

It is not always possible to consider implicit discourse relations simply as explicit relations with the connective removed.

(3) I want to go to New York, but I already booked a flight. (I am not going to New York.)

Type	Frequency	Ratio
Explicit	14722	45.25%
Implicit	13156	40.44%
EntRel	4133	12.7%
AltLex	524	1.61%

Table 2.1: Discourse relation type distribution.

TEMPORAL	CONTINGENCY	COMPARISON	EXPANSION
Asynchronous Synchronous	Cause Condition	Concession Contrast	Alternative Conjunction Exception Instantiation List Restatement

Table 2.2: Sense hierarchy

(4) I want to go to New York, so I already booked a flight. (I am going to New York.)

Here, we have either a COMPARISON.Contrast, or a CONTINGENCY.Cause relation depending on what form the sense takes. If we were to remove the connective token, our linguistic intuition tells us to default for the meaning of Example 4, that is, CONTINGENCY.Cause. How can we use this to our advantage when classifying implicit connectives? (Honest question, still don't know.)

2.1 Related work

2.2 The CoNLL-2016 Shared Task on Shallow Discourse Parsing

At CoNLL 2015 they introduced a shared task on *Shallow Discourse Parsing*. In this task, a system is fed a piece of newswire text as input and returns discourse relations similar to our previous examples. Such a system needs to locate both explicit or implicit discourse relations, identify the spans of its two relations, and finally predict the sense of the discourse connective.

The shared task for the CoNLL 2016 continues upon the work of last year's shared task while introducing Chinese as an alternative evaluation language. A new component is the subtask of *sense classification* where correct argument dependencies and connective are already given, leaving out the sense of the discourse connective. The purpose of this is to allow more focus to be put into solely studying the properties of discourse connectives without having to worry about the pure parsing.

The work in this thesis is built generally upon the CoNLL-2016 shared task, and particularly on the subtask of sense classification.

2.2.1 Findings of CoNLL-2016 Shared Task

The submitted systems for CoNLL 2016 saw a clear trend toward neural network based architectures.

3 Data

For training we will be using the Penn Discourse Treebank (PDTB). PDTB follows the lexically grounded predicate-argument approach as proposed in Webber (2004). It covers the subset containing Wall Street Journal articles from the Penn Treebank, making up approximately one million tokens. When a connective explicitly appears, it will be syntactically connected to the *Arg2* argument of the discourse structure. *Arg1* is the other one. Due to *Arg2* being syntactically bounded to connective, it is easy to automatically classify *Arg2*.

PDTB annotates each structure with types of discourse relations according to a three level hierarchy as seen in Table 2.2, where the first level is made up of four classes: TEMPORAL, CONTINGENCY, COMPARISON, EXPANSION. Each class has a non-obligatory second level of in total 16 types to provide a more fine-grained classification. Due to the third level being considered too fine-grained, it is ignored in this work.

Furthermore, we have some additional resources at our disposal:

- Brown clusters from the RCV1 corpus.
- MPQA subjectivity lexicon.
- Skip-gram Neural Word Embeddings trained on 100 billion words from the Google News dataset.
- VerbNet 3.2.

3.1 Report (2016-02-01)

I want to focus on sense classification, mostly due to the fact that this is provided as a separate task with provided training and test data. If there is time left, I could implement this module into the previous year's best performing end-to-end discourse parser. After having studied the previous year's systems, I have come to the conclusion that there is a gap in representation learning approaches. While I probably will not be alone in this decision (just look at last year's EMNLP), I might be able to contribute with some non-obvious ideas.

There were only two systems that used semantic representations, both from University of Dublin, and both performing relatively weakly in the task. The other submissions are only using ”shallow” features. Okita et al. (2015) focuses on using paragraph vectors along with some additional features to improve sense classification. They obtain reasonable F-score for explicit senses (.88), while the implicit sense classification is quite bad (.11). The implicit sense classification is mainly bad due

senses	connective_token	ratio
(Expansion.Conjunction,)	7355	0.226064
(Contingency.Cause,)	4969	0.152728
(Comparison.Contrast,)	4521	0.138958
(EntRel,)	4133	0.127032
(Expansion.Restatement,)	2640	0.081143
(Temporal.Asynchronous,)	2044	0.062825
(Expansion.Instantiation,)	1392	0.042785
(Comparison.Concession,)	1268	0.038973
(Contingency.Condition,)	1114	0.034240
(Temporal.Synchrony,)	985	0.030275
(Comparison,)	484	0.014876
(Expansion.Alternative,)	435	0.013370
(Temporal.Asynchronous, Contingency.Cause)	148	0.004549
(Temporal.Synchrony, Contingency.Cause)	113	0.003473
(Temporal.Synchrony, Expansion.Conjunction)	109	0.003350
(Expansion.Conjunction, Contingency.Cause)	106	0.003258
(Expansion,)	96	0.002951
(Expansion.Conjunction, Temporal.Synchrony)	92	0.002828
(Temporal.Synchrony, Comparison.Contrast)	51	0.001568
(Expansion.Conjunction, Comparison.Contrast)	47	0.001445

Table 3.1: Frequency of sense labels in training data. Right now connective_token is simply the frequency. I should change this name. Ratio is the frequency ratio.

to low recall (.15), while the precision is on par with explicit sense classification. Furthermore, they find that the classification task struggles with classes with explicit connectives where the surface form is occurring in more than one class. The second paper by Wang et al. (2015) is not focusing on sense classification and thus not provide much detail about its results there. Overall, its results are somewhat disappointing.

Beyond last year's submissions, Ji and Eisenstein (2014) has implemented a parser using representation learning for the RST treebank which uses another discourse framework compared with PDTB. This one shows significant improvements compared to previous SOTA on the treebanks, but the results are in no way comparable with the PDTB. Still, it is the first paper (that I could find) that uses representation learning in a discourse parsing context, and it does so successfully which makes me want to at least look into their manually selected features later on. I am honestly somewhat surprised that none of the other systems looked into using this for the task, especially since the authors of the overview papers considered its results interesting enough to briefly mention it.

What I am most excited about is the idea of applying findings from architectures for the Stanford Natural Language Inference Corpus (Bowman et al., 2015). This is a classification task that I consider to be fairly similar to the sense classification task, and there are several interesting model implementations with strong results I want to further explore. Their website[1] especially mentions Mou et al. (2015) as a particularly

[1]http://nlp.stanford.edu/projects/snli/

efficient recursive neural network model with strong results. Cheng et al. (2016) is a LSTM network model, albeit not as efficient, but receives the highest score so far. Unfortunately, neither of these seem to have published their code which would mean some reimplementation efforts. I don't mind since I would like to learn TensorFlow anyway, and Google's Mat Kelcey has already some code I could work off of, albeit the performance is not as good[2].

That said, there are some things to keep in mind. First of all, neural network approaches are mainly known to be good when there are a lot of training data. That is not necessarily the case here. If the approaches above fail, I will look into if it is possible to use other, more sparse data friendly, architectures with the semantic representations. Also, Braud and Denis (2015) compared shallow features with some dense vector representations and found that shallow features still receive SOTA for implicit discourse relation identification. They do not seem to base any of their findings on word2vec vectors though, which is the kind of features that is allowed in the closed shared task. Also, their concatenation methods differ from the kind of architectures I have in mind. I will have to read this paper more carefully, and see what kind of dead ends I might be able to avoid.

Last minute update: I just found out about Zhang et al. (2015) from EMNLP 2015 which is a shallow convolutional neural network which reaches SOTA on implicit discourse recognition (which is not entirely the same task) on PDTB, making some of the previous worries about neural network approaches less worrisome. I am going to get a good understanding of their approach, read up on the NLI papers, and see how I can combine their efforts.

But first things first: I need to set up a baseline. Here are some suggestions:

- Try to break out the sense classifier from previous year's best performing paper (Wang and Lan, 2015). They use two separate classifiers for explicit and non-explicit senses, based on the original architecture in Lin et al. (2014), while adding a few extra features to boost the performance. The code is freely available, so it should be doable. Although it is unclear if this is actually the best model for sense classification.
- Do the same thing, but for the original architecture. (Lin et al., 2014). The code base is also available.
- Try to figure out which of the systems perform best on
- Implement some other simple system, e.g. an SVM bag-of-words model, potentially with brown clusters due to them

I'm leaning towards the having both the first suggestion, using Wang and Lan (2015), as well as the SVM. The first suggestion due to me wanting to potentially incorporate my classifier into their system, and SVM for its usage as baseline in Zhang et al. (2015) that I just found.

3.1.1 Questions!

- Is this a good plan?

[2]`https://github.com/matpalm/snli_nn_tf`

- Is using Wang and Lan (2015) as a baseline a good idea?
- Am I fooling myself by believing that I can learn anything from studying NLI architectures?
- The sense classes are rather imbalanced (see Table 3). Do you know of any general methods on how to deal with this?
- What am I missing?

3.2 Report (2016-02-25)

Since last time, I have been focusing on three things:

- Learning theory behind neural networks, convolutional NN, recurrent NN, and recursive NN.
- Learning tensorflow
- Setting up experimental architecture.

The first point has been going well. I have a sort of reasonable understanding of the maths behind it (backprop is still magic). I have a good understanding of the theory behind the Tree-Based Convolutional Neural Network (TB-CNN) as described in Mou et al. (2015). I found this particular model interesting due to its incorporation of tree structures into a (for neural networks) easily trainable CNN architecture while receiving very strong results on the NLI task. I haven't looked quite as closely to the other candidates yet.

For the rest to make sense, I will have to make a short summary of how TB-CNN differs from common CNN architectures. Basically, they have incorporated a tree-structure into the convolutional operation. While a common CNN in NLP applications (such as Kim (2014)) runs its convolution on *sequentially* adjacent word features, TB-CNN runs its convolution on *hierarchically* adjacent word features according to its dependency tree. Moreover, while a common CNN has one weight matrix per feature map which is used across all words, TB-CNN has one weight-matrix per *dependency label*.

After going through the NN theory, I started to go deep into Tensorflow which has been fairly straight-forward. This is when I started to realize that implementing TB-CNN will be more difficult than I first imagined, due to the unorthodox convolutional operations in the model which aren't easily supported by neither Tensorflow nor Theano (which are the two frameworks I feel comfortable working with). I still haven't completely given up on being able to implement it, I find its incorporation of tree structures in CNN very interesting and intuitive, but currently I think my time is better spent implementing simpler models.

I have the following models in mind for this:

- A simple logistic regression model just as a proof of concept that my architecture works. This is almost done (see below).
- An SVM to have my baseline somewhat ready. This should be quick.

- Incorporating a feed-forward NN model that has been implemented and released for the NLI corpus, with reasonable results. This should be quick.
- A very successful NN model for sentiment analysis is the Recursive Neural Network by Socher et al. (2013). This, similarly to the TB-CNN incorporates tree structures into its semantic composition, while actually having a Theano implementation for me to work from.
- Cheng et al. (2016) is an LSTM model that currently has the best score for the NLI corpus. Despite having the best score and seeming quite implementable, due to the long training times for these types of model I am somewhat doubtful that I would like to work with them. It is still worth considering, though.
- And just for completion, Kim (2014) is a simple CNN model for sentence processing. This is attractive due to being relatively efficient to train.

To put this work into some context, I will describe my experimental architecture that I have implemented this last week.

3.2.1 Experimental architecture

In the planning and subsequent implementation of the architecture, I focused on having a very modular design, making it easy to plug in and try out new models, features, and different training data. For this to work, each module need to be agnostic about what the others do, and simply expect to receive and output data according to a standardized API. This resulted in the following modules:

extractors

Implements a number of feature extraction models. It takes sentences as input and transforms them into e.g. word2vec features, brown clusters, etc. Outputs a feature matrix (words x word_features).

resources

Takes care of training/test/dev data by reading data and massaging it into adhering to a standardized API. It outputs a feature tensor (sentences x words x word_features) given a set of extractors.

models

Takes a feature tensor as input, reshapes the tensor into its required shape, and trains a model on this. It thereafter saves the model, and tests it given dev or test data. This makes implementing new models fairly easy. Currently, a logistic regression model is at least trainable.

test_report

Saves results along with its configuration and training model in a transparent way. This isn't built yet, but I need some sort of structure to make sense of the hundreds of different configuration permutations I'm going to have. The experimental framework

Sacred[3] seem to promise to do just this in an automated fashion, and I will probably try to incorporate it into my architecture.

At the time of writing, everything is implemented except for testing the trained model, and test_report.py. I hope to finish this in a few days and thereafter focus on implementing models. All sorts of parameters and hyperparameters are collected in one big config file.

3.2.2 Things to ponder

One big difference between the NLI task and this task is the number of output classes (see table 2.2). While NLI has 2 different classes, PDTB has 16. Furthermore, some instances have *two* concurrent classes, that is, an instance can have e.g. *CONTINGENCY* and *Temporal.Synchrony* at the same time. If we were to treat all of these as atomic units, we would have 56 unique classes to predict. Most of these dual-class instances are very rare and I am thinking about ignoring these for now. Furthermore, I am also thinking about focusing on getting the top-level hierarchy right for now, that is, only classifying *TEMPORAL*, *CONTINGENCY*, *COMPARISON*, and *EXPANSION*, and look into the second level hierarchy when I have a working model for the first hierachy. I think splitting up the classification into two separate hierarchies to be beneficial for accuracy as well.

I am still uncertain how I can make use of the implicit/explicit relations. Obviously explicit gives me an extra feature which helps a lot, but I am uncertain if it is best to have an entirely separate classifier for implicit relations, or treat them as the explicit relations with a specific IMPLICIT feature in place of its explicit connective.

Another thing I wonder is: can I have access to a GPU to run my experiments on? It would be nice to be able to speed up the tuning a bit.

I think that's it for now!

3.3 Report (2016-03-22)

This will only be a quick description of my two current models with some initial results.

Word vectors below refers to either randomized fixed-sized vectors, or pre-trained word2vec.

Each instance has the following arguments:

- connective_token: Refers to the binding tokens between the two arguments. This can be null if instance is of type Implicit (underlined).
- arg2_text: Refers to the text string which is syntactically binded with connective_token (bolded).
- arg1_text: Refers to the other argument (cursive).

(5) *Boeing would make cost-of-living adjustments projected to be 5 for each year of the contract* <u>though</u> **the union has called the offer insulting**.

[3] http://sacred.readthedocs.org/en/latest/

3.3.1 SVM Baseline

The SVM is a simple model, doing the following:

1. Generate word vectors for all words for the training instance's arguments.
2. Calculate the mean word vector (centroid) for each argument.
3. Concatenate the arguments' centroid vectors.
4. Train the SVM classifier on this feature vector.

I'm using the default parameters as set by scikit-learn. This include a non-linear kernel, RBF:

```
    SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)
```

3.3.2 CNN Model

The CNN model is based on the tutorial *Implementing a CNN for Text Classification in TensorFlow*[4]. This implements a model very similar to Kim (2014), which is a text classification model using convolutional neural networks with word vectors for sentence-level classiciation tasks. It learns task-specific vectors by tuning the word vectors during training, and at the time gave SOTA results for a number of sentiment analysis tasks. See 3.3.2.

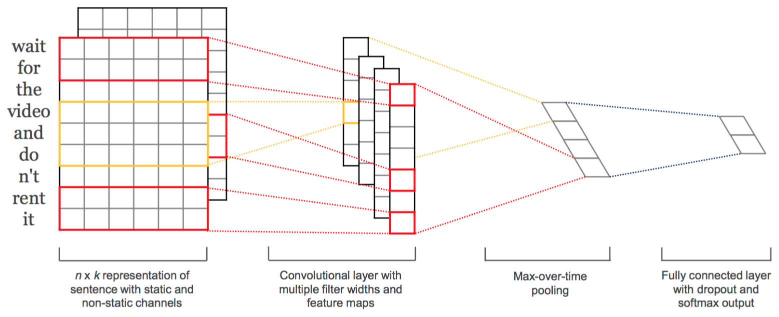
wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: Model architecture with two channels for an example sentence.

The model requires a fixed number of words for training per argument. Each argument, as listed above, therefore has a *max_words* setting, which is currently set to 5 for connective tokens and 25 for each argument text. If the argument is shorter than the max setting, we pad the sentence with additional words. If it's longer, we cut the sentence at the given point. Each arguments instance matrix are concatenated, making the input matrix dimensionality (25+25+5, embedding_dimensionality).

This too has a number of tunable features that I have yet to look closer into. The two most prominent are the number of convolutional filters, as well as the filter sizes.

[4] http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

A filter size determines the span of the convolutional operation, e.g, a filter size of 2 looks at bigrams. Each filter is randomly initialized, and will through training tune into finding different features in the sentences. I also plan on experimenting with different number of convolutional layers with different pooling algorithms.

```
embedding_dim: 300
filter_sizes: [1,2]
num_filters: 128
dropout_keep_prob: 0.5
l2_reg_lambda: 3
batch_size: 128
num_epochs: 100
evaluate_every: 100
checkpoint_every: 100
allow_soft_placement: true
log_device_placement: false
max_words_in_sentence: 25
```

3.3.3 Results so far

So far, the SVM is doing better, although CN seems to be somewhat better at classifying implicit relations. The reason for this is that when the connective is not present, it defaults to the class EntRel which is neither explicit nor implicit, while the CNN behaves more randomly. I haven't looked into what kind of results randomly assigned labels would give, so I'm not sure it actually beats random yet.

I'm going to read up on other CNN models to get some tips on how I can best structure the input (perhaps split up each argument input into separate convolutional networks which are then merged. I think I saw someone doing something like this, but I have had trouble finding it.). Somewhat surprisingly, randomized vectors actually performs *better* than word2vec. If this has to do with randomized vectors being more easily tuned, or if there is a bug somewhere, I'm not sure. I will have to look into this further.

3.3.4 Todo

Whether or not I will send something into the shared task is depended on whether I can make this model work. I won't have time for a lot of tuning, so I will try to read as much as possible on different convolutional neural network implementations to get a good overview of what parameters seem to work and what I should think of.

Bibliography

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Chloé Braud and Pascal Denis. Comparing word representations for implicit discourse relation classification. In *Empirical Methods in Natural Language Processing (EMNLP 2015)*, 2015.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long Short-Term Memory-Networks for Machine Reading. *arXiv preprint arXiv:1601.06733*, 2016.

Yangfeng Ji and Jacob Eisenstein. Representation Learning for Text-level Discourse Parsing. In *ACL (1)*, pages 13–24, 2014.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, 2014. Association for Computational Linguistics.

Ziheng Lin, Hwee Tou Ng, and Min-Yen Kan. A PDTB-styled end-to-end discourse parser. *Natural Language Engineering*, 20(02):151–184, 2014.

Lili Mou, Men Rui, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Recognizing Entailment and Contradiction by Tree-based Convolution. *arXiv preprint arXiv:1512.08422*, 2015.

Tsuyoshi Okita, Longyue Wang, and Qun Liu. The DCU Discourse Parser: A Sense Classification Task. *CoNLL 2015*, page 71, 2015.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

Jianxiang Wang and Man Lan. A refined end-to-end discourse parser. *CoNLL 2015*, page 17, 2015.

Longyue Wang, Chris Hokamp, Tsuyoshi Okita, Xiaojun Zhang, and Qun Liu. The dcu discourse parser for connective, argument identification and explicit sense classification. *CoNLL 2015*, page 89, 2015.

Biao Zhang, Jinsong Su, Deyi Xiong, Yaojie Lu, Hong Duan, and Junfeng Yao. Shallow Convolutional Neural Network for Implicit Discourse Relation Recognition. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2230–2235, 2015.