Himanshu Joshi

CS 662 Final report

Project Goal

The goal of this project is to study the compositional properties of word embeddings in aid of sentence parsing. I intend to use word and phrase embeddings already trained by (Mikolov, Chen, Corrado, & Dean, 2013), (Ustun, Rosenbloom, Sagae, & Demski, 2014) and potentially (Vaswani, 2013) for this project. These embeddings will be used to annotate learnt PCFG rules learnt from the database. The parsing algorithm shall be extended with simple heuristics to account for these embeddings.

The success of the project would be both:

- Characterization of the operators needed for obtaining higher order ordered sequential compositions of individual word vectors and phrase based word vectors.
- Improvement in F1 and recall scores of the parses over the baseline CKY parsing algorithm implementation.

Method description

The aim is to explore enhancement of parsing accuracy with the aid of semantic information. I am assuming the grammar is in Chomsky Normal form and the trees are binary (I am not entirely sure if these are reasonable assumptions to make). I shall learn a PCFG from the dataset as a first step and then implement the CKY parsing algorithm to generate parse trees.

The semantic information shall be obtained from sources listed above. Thus there is a distributed representation for each terminal rule's production in the vocabulary. Then using the methods described in (Plate, 1995) vector compositions of each unary rule, for example, the embedding for the non-terminal *'DT'* shall be obtained from the embeddings of *'the'* and *'this'* (in accordance with the rules *DT -> the* and *DT -> this*) using the trace encoding operator described in (Plate, 1995). This turns out to be a simple vector addition of respective word vectors. I tried a variant of this by using weighted composition, where the weights were obtained from the PCFG.

Similarly, for example, the vector for *NP** shall be obtained by using the composition operator on the embeddings of the non-terminals '*DT*' and '*NNP*' (corresponding to *NP* -> DT NNP*). Thus we have semantic vector embeddings for all terminals(T) and non-terminals (NT) in the grammar and vocabulary $\{T_{emb}, NT_{emb}\}$. The unary rule heads such as DT do not contain recursion, where as the binary rule heads such as NP can contain recursion. To handle recursion, I was going to obtain an intermediate representation of the rule head using the non-recursive rules of that rule head. Then using this intermediate representation, I can obtain a recursive representation by using it on the

RHS of the rule.[1] For example, consider the NP rules: $NP \rightarrow DT\ NN$ and $NP \rightarrow NP\ NN$. I can use the first rule to obtain NP_1 and use that in the first rule to obtain $NP_2 \rightarrow NP_1 NN$ and finally, to obtain the PoS embedding for NP, we use $NP_2 = NP_1 + NP_2$ (or a weighted version of this, where the weights correspond to the probabilities in the PCFG for respective rules).

Here are two simple heuristics that I can add:

- When an unseen/unknown word is encountered, find the unary class closest to it, and treat it as a word of that class and generate parse accordingly. Alternatively, use a probabilistic ranking of each unary rule class for the unseen word, based on it's distance to the unary PoS embedding.
    - I pushed this method to see if I can generate the whole parse using nothing but the word embeddings for unary rules and using the binary rules from the PCFG. The results were interesting, though not comparable with the PCFG method.

- When a parse is not generated, we find the class semantically closest to this subtree and use that class to represent this subtree.

Due to time constraints, I was able to implement the first heuristic only, i.e. using word embeddings to obtain PoS embeddings for unary rule heads. Instead of returning the 1 best class from its semantic embedding, I converted the similarity measure obtained from the distance of that word to each class into a probability measure[2]

$$P(class|word) = \frac{\exp\left(\cos\left(word, class\right)\right)}{\sum_{class'} \exp(\cos(word, class'))}$$

Data description

I used the dataset from HW4, which is the ATIS subset of the Penn Tree Databank. This will be my one-hour baseline. I assume I shall have binary rules that can generate grammar in Chomsky Normal form. I used the word vectors from https://code.google.com/p/word2vec/ .

Evaluation method

- F1 and recall score comparison of baseline CKY implementation and enhanced/modified CKY implementation.

---

[1] Paul suggested this particular iterative method. I was going to use the intermediate result with a random projection instead of using it directly.

[2] Strictly speaking cosine distance measure cannot be directly used as a probability measure, as it can be negative. I used the exponential function to convert this to a positive value.

- Evaluation of the influence of different forms of embedding compositions on the CKY parser.
    - This should constitute comparing performance on CKY using both commutative and non-commutative operators.
        - I did not implement the binary PoS embeddings and did not try the commutative & non-commutative operators. I did try the weighted and unweighted composition for unary rule heads.
    - Performance of various heuristics that I might try.
    - I will also evaluate embeddings obtained from different sources.

Baseline method

The baseline method was the CKY parser implemented for HW4. Due to the fact that Word Embedding dataset was too large and corresponding word2vec routines available in C, I implemented my HW4 parser in C. This reduced the computation time by a decent amount, although wall clock time did not reduce by more than 25% (mostly because Java seems to be doing a decent job at multithreading).

Results

Due to time constraints, I was able to implement heuristic 1 only. In the simplest case, I returned the nearest PoS class for each unseen word, with results shown in row 2. This was improved upon by returning a probabilistic measure for all classes given this word. This is shown in row 3, and this is comparable to what HW4 does with the method of using *<unknown>* for each unseen word. In row 4, I used a weighted variant of the vector addition but with the same result as without weights. Testing on a larger dataset might help differentiate performance between the two. Finally, I tried to push this method to see how well it will do if I use only PoS embeddings for every word instead of using PCFG rules. Although the results are not very good, more sophisticated training methods that improve the syntactic quality of the word vectors can help (one such approach might be to train a syntactic classifier using these word vectors as starting points).

Method	F1	Recall	Precision
CKY Parser HW4	0.8800	0.84388	0.9195
Unary PoS Embeddings for unseen words (1 Best PoS class)	0.7663	0.6434	0.9472
Unary PoS Embeddings for unseen words (returns all 57 classes)	0.88571	0.85021	0.92431
Unary PoS embeddings with weighted composition for unseen words (all 57 classes)	0.88571	0.85021	0.92431
Unary PoS Embeddings for all words (all 57 classes)	0.5962	0.6075	0.58365

Table 1 Comparison with baseline.

Himanshu Joshi

Discussion about results

- The HW 4 baseline used a heuristic to keep aside some probability mass of each unary PoS class to generate and *<unknown>* word. When using PoS Embeddings, I did not use such a default clause. Instead, I return a probability score based on the current word's embedding to all PoS classes. Thus, depending on the exact embedding of the word and the quality of the composition, there was no guarantee I will match the results in my HW4 baseline.
    - Thus, I was able to show that semantic information can aid syntactic parsing in a limited way.
    - With the current dataset, there is not much to choose between two method. Below, in Table 2, I have reviewed the performance on unknown words as handled by the HW4 method and the distributed vector implementation of the parser.
    - One disadvantage of the distributed method is that it requires a lot of processing power to train the vectors. Whereas previously, my entire application fit into 20MB of main memory, with just the word2vec subset needed for the parser (a total of 498 words), I needed 321 MB in the main memory.

- There was no effect of weighing the vectors before composition. This could be a limitation of the dataset itself. Perhaps more data in the PCFG might help.

- Using nothing but the unary PoS embeddings for terminal rules and using the PCFG rules for binary rules, I was able to obtain a non-trivial measure of F1, recall and precision. Although it is nowhere near the HW4 results obtained via PCFG, being able to obtain non-trivial measures seems to be an indicator that semantic information can help in parsing, if we have ways of improving the semantic vector information. We have a long way to go before a parse can be obtained simply by using nothing but word vectors.

- The details of each unseen word and its classification as compared to the baseline is shown in Table 2. As can be seen, the semantic method does better on Nouns and Verb tags in comparison with the *<unknown>* method.

Word (occurences)	PoS Embedding classification	Classification using <unk>	Reference classification
breakfast (2)	NN (2)	RB (2)	NN (2)
Are (3)	VBP(2), VBZ(1)	VBP(3)	VBP(3)
Where(1)	WP	WRB	WRB
does(1)	VBP	VBP	VBZ
Alaska(1)	NNP	NNP	NNP

Word (occurences)	PoS Embedding classification	Classification using <unk>	Reference classification
eighty(1)	CD	CD	CD
seventh(1)	RB	RB	JJ
provides(1)	VBP	IN	VBZ
only(1)	IN	IN	RB
connecting(1)	DT	JJ	VBG
M(1)*	-	-	SYM
anywhere(1)	NNP	NNP	RB
Display(1)	WDT	WDT	VB
within(1)	IN	RB	IN
each(1)	DT	DT	DT
Florida(1)	NNP	NNP	NNP
Restriction(1)	DT	NN	JJ
slash(1)	IN	IN	SYM
restrictions(1)	NN	NN	NNS
Correct	36%	36%	100%

Table 2: Detailed comparison of the <unk> method with heuristic 1 (unary PoS embeddings used for unseen words). * indicates failed to generate a parse.

Follow up experiments

Several follow up experiments can be performed. The obvious ones are:

- Testing this approach on a larger dataset. One that has more unknown words in the test set.
- Implementing the composition operator to obtain binary PoS embeddings.
- The other would be to improve the quality of the existing unary PoS emebddings by improving the word vectors via some form of supervised classification using the word vectors.

After, the first item, I am undecided as to which to pursue first.

- The reason why some parses fail at a higher level is that the PCFG is too rigid and is unable to make new rules on the fly, for example parse for "What is M I A ?" fails because such rule to combine two SYMs does not exist: $NP \rightarrow SYM\ SYM$. Thus, we could add that flexibility via the cosine similarity to probability metric

used for unary rules. This might generate higher scores (and also generate a LOT more work for the parser, please refer to the Appendix to get a rough idea).

- If using the existing binary PCFG rules and unary PoS embeddings but no unary PCFG rules, the f1, precision and recall metrics are not very good (last row in Table 1). So perhaps the next step after testing on larger dataset should be to obtain better quality word vectors that retain more syntactic information.

Code

The code is available at:
https://hima_cogarch@bitbucket.org/hima_cogarch/pcfg_dvrs.git or
git@bitbucket.org:hima_cogarch/pcfg_dvrs.git

I have been regularly running XCode's version of lint to ensure there are no warnings/lint errors. Towards the last few days of the project, I was a bit sloppy and that shows in grammar.c (and a few lint errors remain in grammar.c).

Appendix

I. How to run:
Unzip and cd to the folder. Type make to make the project and run with './cky_dvrs '
The project can also be compiled and run in XCode.

II. Performance comparison:

1. HW4 baseline performance in Java:

```
Sentence length vs runtime(ns):
2 22763.0
3 62933.5
4 250965.5
5 426537.3333333333
6 9550338.444444444
7 1169825.25
8 8100829.3
9 2423590.2
10 9370926.625
11 1.2303101E7
12 2.48341742E8
13 3904586.0
14 1.8272089333333332E7
21 1.2430446E7
```

2. HW4 baseline performance in C:
Total rules parsed 752
Parse chart initialization, num cells allocated : 1024, inited: 1024
Total number of words : 498 of size 300
Running cky parser on set : ./data/dev.strings
.....

Sentence length vs runtime (ns)
cps : 1000000 , cpns : 0.001000
2 : 0.000000 ns
3 : 0.000000 ns
4 : 0.000000 ns
5 : 0.000000 ns
6 : 111.111111 ns
7 : 500.000000 ns
8 : 200.000000 ns
9 : 400.000000 ns
10 : 375.000000 ns
11 : 1000.000000 ns
12 : 0.000000 ns
13 : 0.000000 ns
14 : 3000.000000 ns
21 : 1000.000000 ns
CKY parser result in ./debug/dev.parsed

3. Degradation of performance with unary PoS embeddings for unseen words.

Total rules parsed 752
Parse chart initialization, num cells allocated : 1024, inited: 1024
Total number of words : 498 of size 300
Running cky parser on set : ./data/dev.strings
.....
Sentence length vs runtime (ns)

cps : 1000000 , cpns : 0.001000
2 : 0.000000 ns
3 : 0.000000 ns
4 : 0.000000 ns
5 : 111.111111 ns
6 : 1666.666667 ns
7 : 0.000000 ns
8 : 1900.000000 ns
9 : 600.000000 ns
10 : 2375.000000 ns
11 : 3000.000000 ns
12 : 76000.000000 ns
13 : 0.000000 ns
14 : 3333.333333 ns
21 : 1000.000000 ns
CKY parser result in ./debug/dev.parsed

4. Degradation of performance with unary PoS embeddings for all words.

```
Total rules parsed 752
Parse chart initialization, num cells allocated : 1024, inited: 1024
Total number of words : 498 of size 300
Running cky parser on set : ./data/dev.strings
.....
Sentence length vs runtime (ns)

 cps : 1000000 , cpns : 0.001000
 2 : 0.000000 ns
 3 : 5000.000000 ns
 4 : 28000.000000 ns
 5 : 81555.555556 ns
 6 : 230555.555556 ns
 7 : 501250.000000 ns
 8 : 994900.000000 ns
 9 : 1812600.000000 ns
 10 : 2884875.000000 ns
 11 : 5101000.000000 ns
 12 : 5953000.000000 ns
 13 : 12254000.000000 ns
 14 : 18018000.000000 ns
 21 : 150990000.000000 ns
CKY parser result in ./debug/dev.parsed
```

References

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.

Plate, T. A. (1995). Holographic reduced representations. . *IEEE transactions on Neural Networks , 6* (3), 623-641.

Ustun, V., Rosenbloom, P. S., Sagae, K., & Demski, A. (2014). Distributed Vector Representations of Words in the Sigma Cognitive Architecture. Vaswani, A. Z. (2013).

Decoding with Large-Scale Neural Language Models Improves Translation. . *In EMNLP* , 1387-1392.