Overview

- Dependency Model with Valence
    1. Introduction
    2. Implementation
- Thousand random restarts
- Future Work

Dependency Model with Valence

- Unsupervised Dependency Parser
- Conceived by Dan Klein
- Generative model
- Requires only 5,773 sentences to train juxtappose state of the art supervised parsers which need around 40,000 annotated sentences

Example

John saw Mary

Generate the root

root John saw Mary

Repeat the next few steps recursively

For a given word, generate all the right dependents

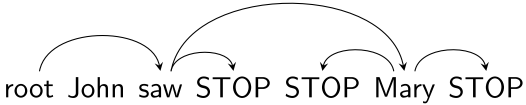
root John saw Mary

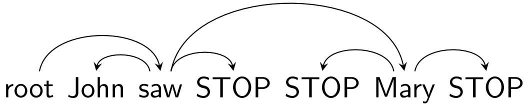
When a word no longer takes any dependents on the right, generate STOP symbol to the right

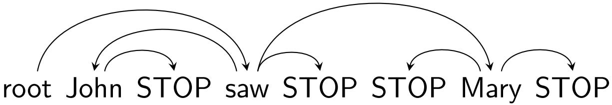
root John saw Mary STOP

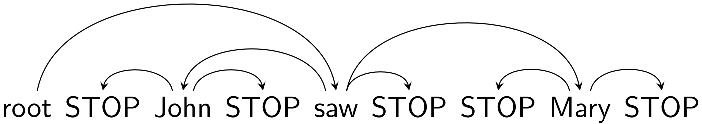
Generate all the left dependents of the word. When a word no longer takes any dependents on the left, generate STOP symbol to the left

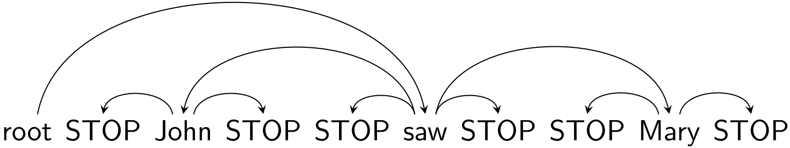
root John saw STOP Mary STOP


root John saw STOP STOP Mary STOP


root John saw STOP STOP Mary STOP


root John STOP saw STOP STOP Mary STOP


root STOP John STOP saw STOP STOP Mary STOP


root STOP John STOP STOP saw STOP STOP Mary STOP

$$P(D(h)) = \prod_{dir \in (l,r)} \prod_{a \in deps_D(h,dir)} P_{STOP}(\neg STOP|h, dir, adj)$$

$$P_{CHOOSE}(a|h, dir)P(D(a))P_{STOP}(STOP|h, dir, adj)$$

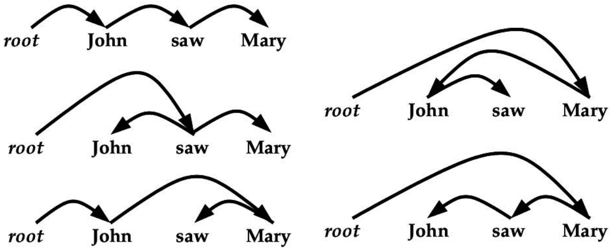
Goal of the Parser

To determine the parameters for underlying distribution:

- The probability that a head word takes a modifier (depProb[h, m, direction])
- The probability that a head word continues to take further arguments (contProb[h, direction, adj])
- The probability that a head word stops taking further arguments (stopProb[h, direction, adj])

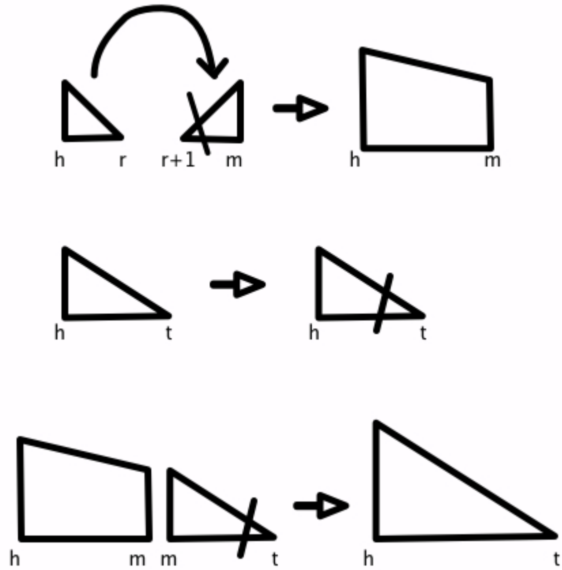
Implementation of DMV

A sentence can have several possible parses:



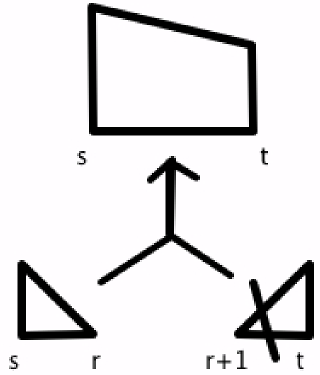
Eisners parsing algorithm – keep track of the counts of all possible parses.
A hypergraph – efficient data structure – calculating the probabilities of each one of these parses.

Modified Eisner's parsing algorithm

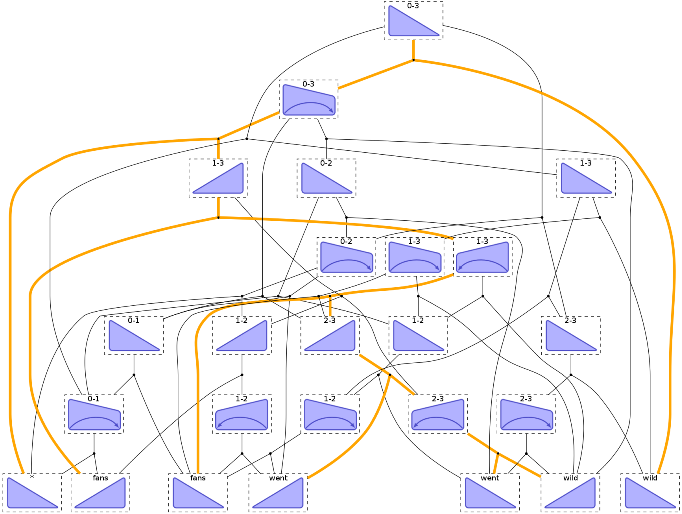


Hypergraph

A hypergraph $\mathrm{H} = (V, E)$, where V is the set of vertices, E is the set of hyperedges.
Hyperedge $e \in E$ of a weighted graph is a triple $e = (T(e), h(e), f(e))$, where $h(e) \in V$ is its head vertex and $T(e) \in V^*$ is an ordered list of tail vertices.
$f(e)$ is a weight function



s
t
s
r
r+1
t


0-3
0-3
1-3
0-2
1-3
0-2
1-3
1-3
0-1
1-2
2-3
1-2
2-3
0-1
1-2
1-2
2-3
2-3
*
fans
fans
went
went
wild
wild

Algorithm 1 Compute Weights

Lets assume incomplete constituent as i.c and complete constituent stop as c.s

if edge.headNode $\in$ i.c **then**

    return depProb[edge.headWord, edge.modifierWord, edge.dir] $\times$ contProb[edge.headWord, edge.dir, edge.isAdj]

else if edge.headNode $\in$ c.s **then**

    return stopProb[edge.headWord, edge.dir, edge.isAdj]

else

    return 1

end if

Implementation

- The insideOutside algorithm is run on the entire hypergraph.
- The inside probability of the root of the hypergraph gives the total probability of the sentence Z.
- The marginals of the nodes and the edges of the hypergraph are computed using the PyDecode library.
- counts = marginals / Z
- The em algorithm is run 10 times for all the sentences

Related work

- (Smith and Eisner, 2005) used contrastive estimation together with DMV.
- (Cohen et al., 2008) used logistic normal priors
- (HeaddenIII et al., 2009) extended the valence and conditioned generating a new argument on whether it is adjacent or not. $P_{CHOOSE}(a|h, dir)$ in the above equation is thus substituted by $P_{CHOOSE}(a|h, dir, adj)$
- (Spitkovsky et al., 2011) observed a strong connection between English punctuation and phrase boundaries, split sentences at punctuation marks and imposed parsing restrictions over their fragments.

Initialization of DMV

Consider a sentence with words $w_1 \ldots w_n$ where n is the number of words in the sentence.

(1) Each word has a uniform probability of becoming a ROOT.

$$P(ROOT) = \frac{1}{n}$$

(2) The probability of dependency between two words is inversely proportional to the distance between them.

Linguistic intuition that shorter dependencies are preferable to longer

Thousand random restarts

Algorithm 2 EM algorithm for a thousand random restarts

```
for iterations = 1 to 10 do
    for sentence in corpus do
        Build hypergraph
        for multinomial in Multinomials do
            Update counts for sentence. Estimation step
        end for
    end for
    for multinomial in Multinomials do
        Recompute the probabilities. Maximization step
    end for
end for
```

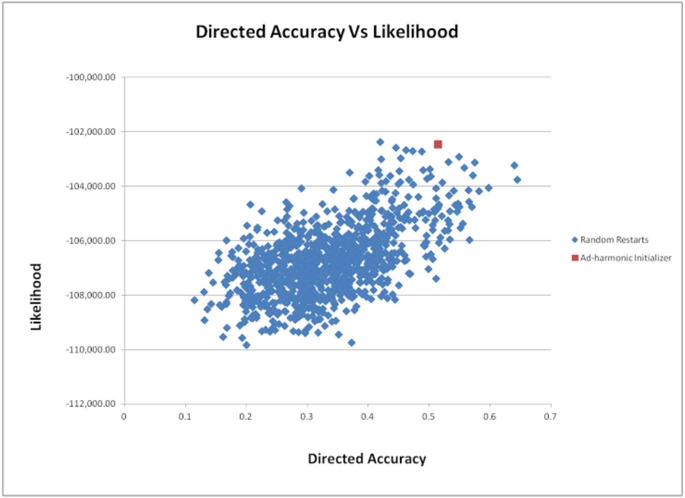


Figure: Directed accuracy vs likelihood

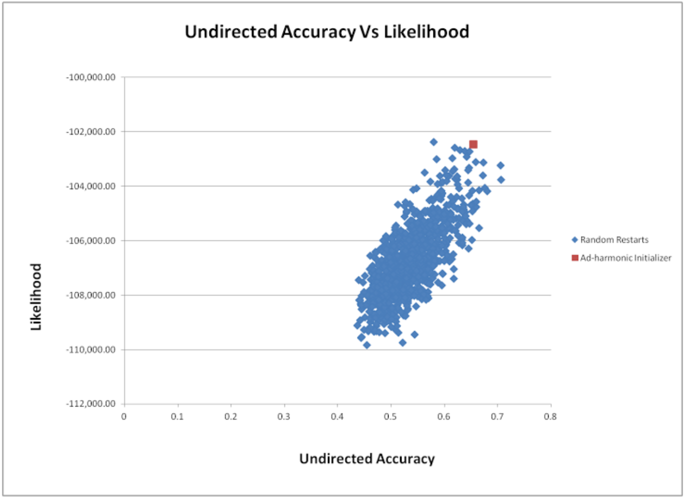


Figure: Undirected accuracy vs likelihood

Characteristic	Ad-harmonic Initializer	Random Initializer
Undirected accuracy	65.5	70.56 (+5.06)
Directed accuracy	51.5	55.59 (+4.09)
Likelihood	-102,453.49	-102,375.79 (-77.7)

Table: Comparing accuracies of Ad-harmonic and Random Initializer

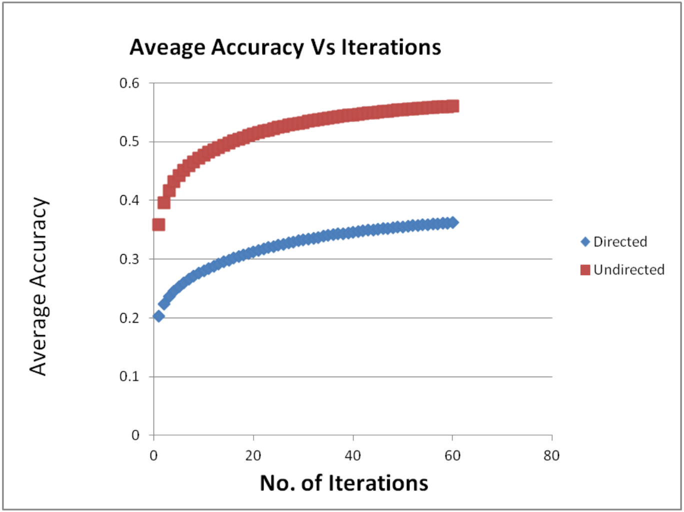


Figure: Average accuracy per iteration

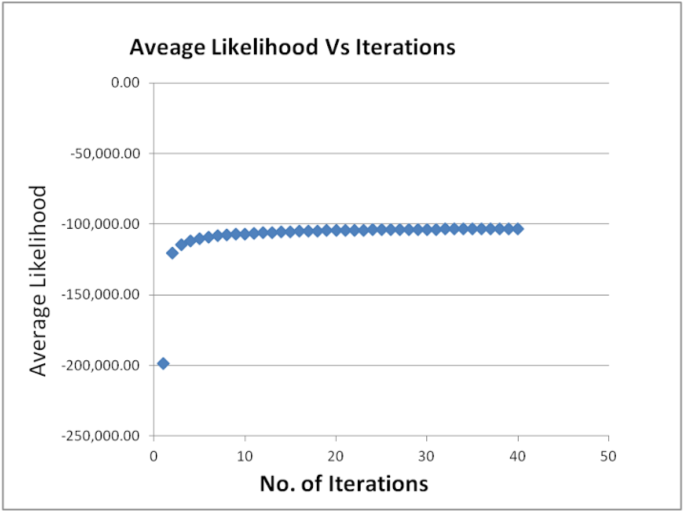


Figure: Average likelihood per iteration

Future Work

- It takes 180 minutes to run EM for 60 iterations with 40 random restarts.
- The time taken to build the model is directly proportional to the number of random restarts
- To scale to a million random restarts, the time taken to build the model must be independent of the number of random restarts.