Statistical Machine Learning and Tree Adjoining Grammar Parsing

Neural Networks in NLP

Jungo Kasai

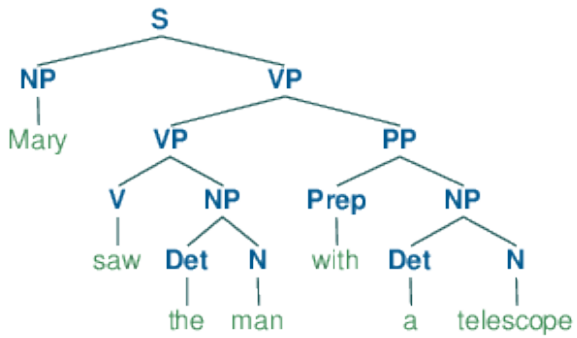
Yale University
Advisor: Robert Frank at the Department of Linguistics at Yale

March 8, 2017

Outline

Parsers and Formalisms

Figure : Mary saw the man with a telescope. [1]



We need ingredients or a constrained set of rules to generate such a tree.

- Context Free Grammar (CFG)
- Dependency Grammar
- **Tree Adjoining Grammar (TAG) (Joshi 1975)**
- Combinatory Categorial Grammar (CCG) (Steedman 1987)

[1] Source: `http://www.cse.chalmers.se/~cajohn/teaching/NLP/HT2016/lectures/1.html`

Context Free Grammar

Example Set of Rules

$$S \rightarrow NP\ VP \quad VP \rightarrow really\ V \quad VP \rightarrow likes$$
$$NP \rightarrow Mary \quad NP \rightarrow John$$

- Insight: Write a set of grammatical rules
- Before 1990: Non-statistical methods scale badly.
- Statistical Machine Learning (Probabilistic Context Free Grammar)
  - The Penn Tree Bank Project (Marcus et al. 1994)

Dependency Grammar

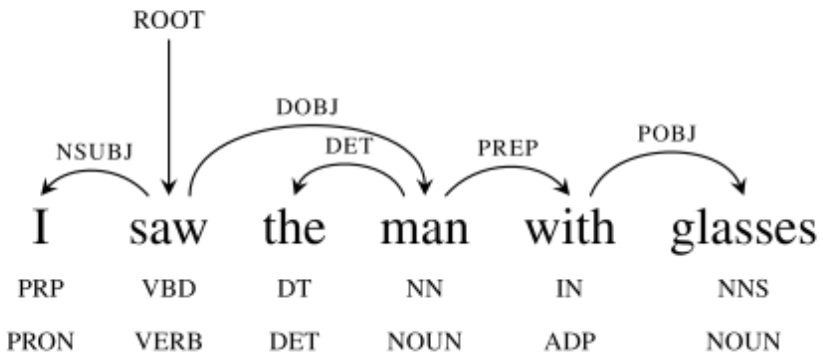


Figure : Dependency Structure of "I saw the man with glasses" [2]

- CYK Algorithm $O(n^3)$
- Transiton-based (Shift-reduced) parsing
  - Deterministic Oracle
  - Malt Parser (Nivre 2003) $O(n)$
  - Parsey McParseface by Google (May 2016)
  - Will be covered in the final presentation.

[2] Source: `http://mlnote.com/2016/12/10/SyntaxNet%20Neural-Models-of-Syntax/`.

Tree Adjoining Grammar

- Decompose the parsing problem into (Joshi and Bangalore 1994):
  - Supertagging (Classification) by the 4500 rich categories (vs 46 parts of speech)
  - Stapling to derive a tree
  - Conjecture: given a perfect supertagger, stapling should be easy; Supertagging is "Almost Parsing"
- Issues and obstacles
  - Classification was not accurate enough
  - Make use of probability distributions over possible supertags
- Applied to bio-informatics
- Midterm Report: Classification or Supertagging
- Final Presentation: Stapling
- Time Permits: Applications of TAG parsing to extrinsic tasks

Tree Adjoining Grammar and Supertagging

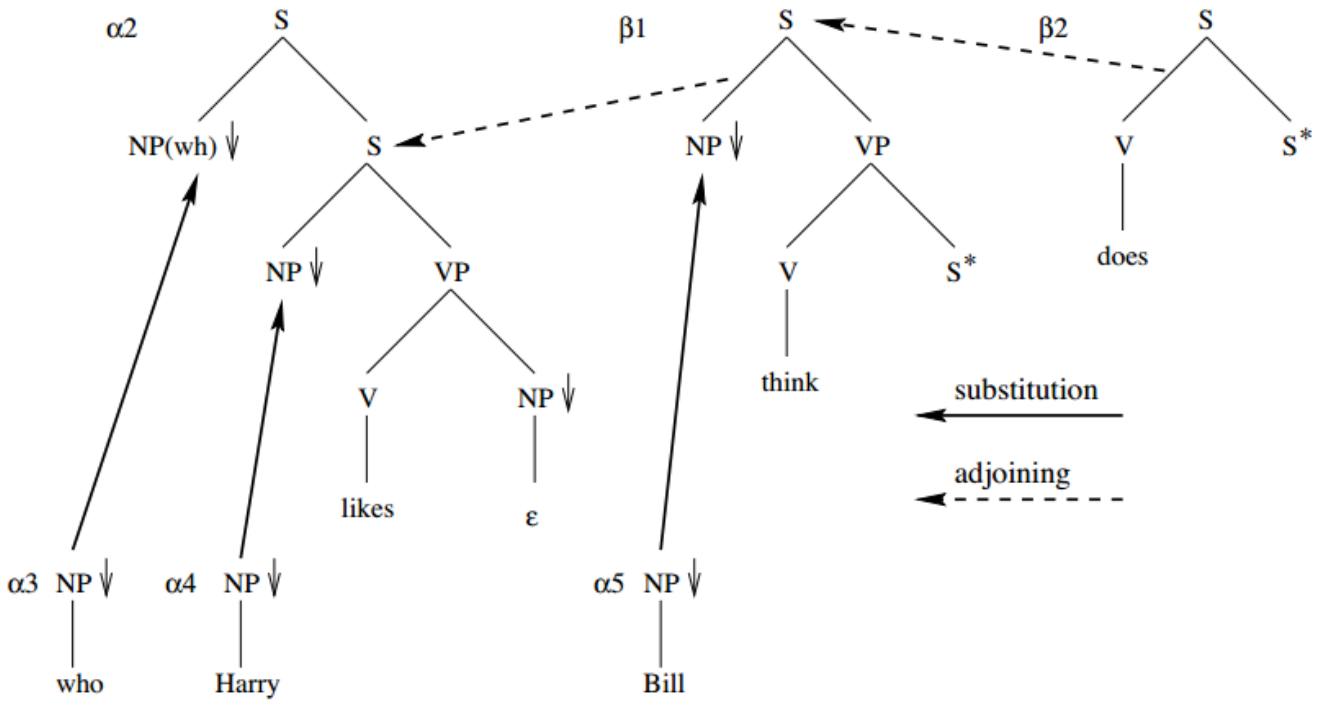


Figure : Derivation of "Who does Bill think Harry likes?" [3].

- 2 types of operations: Substitution, Adjoining

[3] Source: https://muse.jhu.edu/chapter/764336

Tree Adjoining Grammar and Supertagging

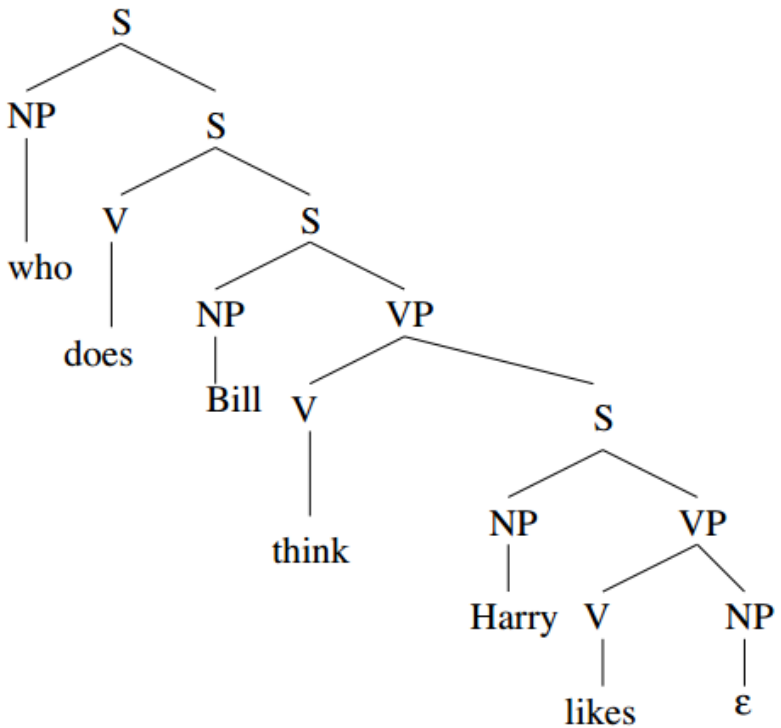


Figure : Derived Tree of "Who does Bill think Harry likes?" [4]

[4] Source: https://muse.jhu.edu/chapter/764336

Tree Adjoining Grammar and Supertagging

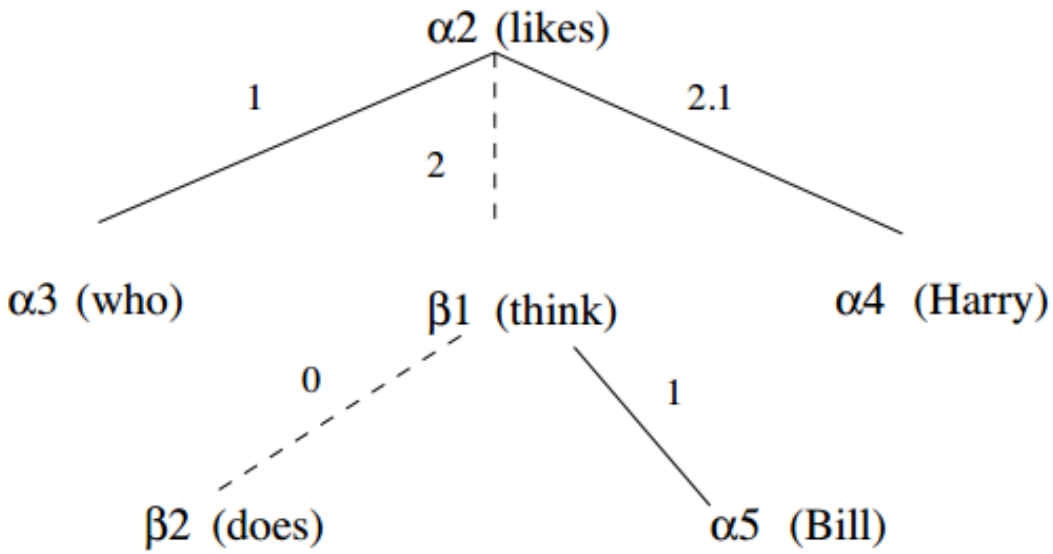


Figure : Derivation Tree of "Who does Bill think Harry likes?" [5].

[5] Source: https://muse.jhu.edu/chapter/764336

Combinatory Categorial Grammar

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{the}{NP/N}\ \dfrac{dog}{N}}{NP}>}{S/(S\backslash NP)}T_{>}\qquad \dfrac{bit}{(S\backslash NP)/NP}}{S/NP}B_{>}\qquad \dfrac{John}{NP}}{S}>$$

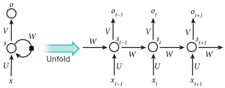
Figure : Derivation of 'The dog bit John' [6].

- Analogous to Lambda Calculus
- 2 types of operations: Composition, Type Raising
- Applied to Question Answering, Information Extraction (Rimell, Clark, and Steedman 2009)

[6] Source: https://en.wikipedia.org/wiki/Combinatory_categorial_grammar

Outline

Recurrent Neural Networks



7

- Simple Recurrent Neural Networks
  - Multiplicative Update
  - $s_{t+1} = Ws_t$
  - Vanishing/Exploding Gradients
- LSTMs
  - Additive Update
  - $s_{t+1} = f_t \odot s_t + g_t \odot i_t$
- Recipe to Apply Neural Nets to NLP
  - NLP vs Computer Vision
    - Sequential Structures, Long Distance Dependency
    - Discrete Inputs $\Longrightarrow$ Word Embedding (Vectorization)
  - Regularization, Generalization

[7] Source http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Word Embedding (Vectorization)

- Typically, 100, 200, or 300 dimensions of real values
- Gets fed to the neural networks
- Hopefully, preserves linguistically sensible structures
- Wordnet (Starting in 1985 at Princeton)

Word Embedding (Vectorization)

- Typically, 100, 200, or 300 dimensions of real values
- Gets fed to the neural networks
- Hopefully, preserves linguistically sensible structures
- Wordnet (Starting in 1985 at Princeton)

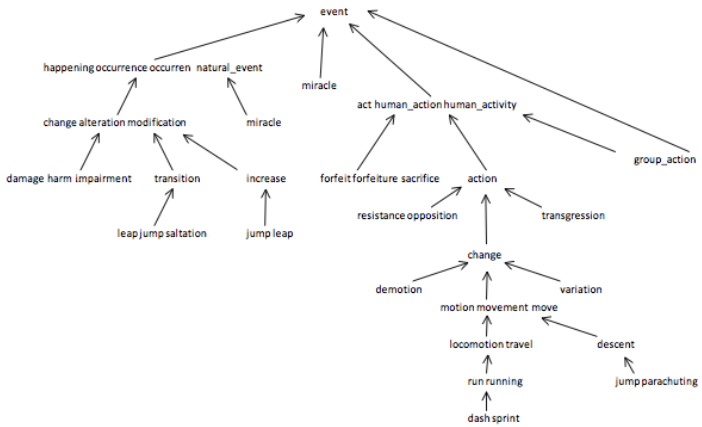


Figure : Source: `http://www.cs.princeton.edu/courses/archive/spring11/cos226/assignments/wordnet.html`

Word Embedding (Vectorization)

- Typically, 100, 200, or 300 dimensions of real values
- Gets fed to the neural networks
- Hopefully, preserves linguistically sensible structures
- Wordnet (Starting in 1985 at Princeton)
- (PMI) Matrix Factorization (SVD, PCA etc)
  - Turney 2001
  - Global Approach
- Word2Vec (Mikolov et al. 2011)
  - CBOW and Skipgram
  - Local Approach

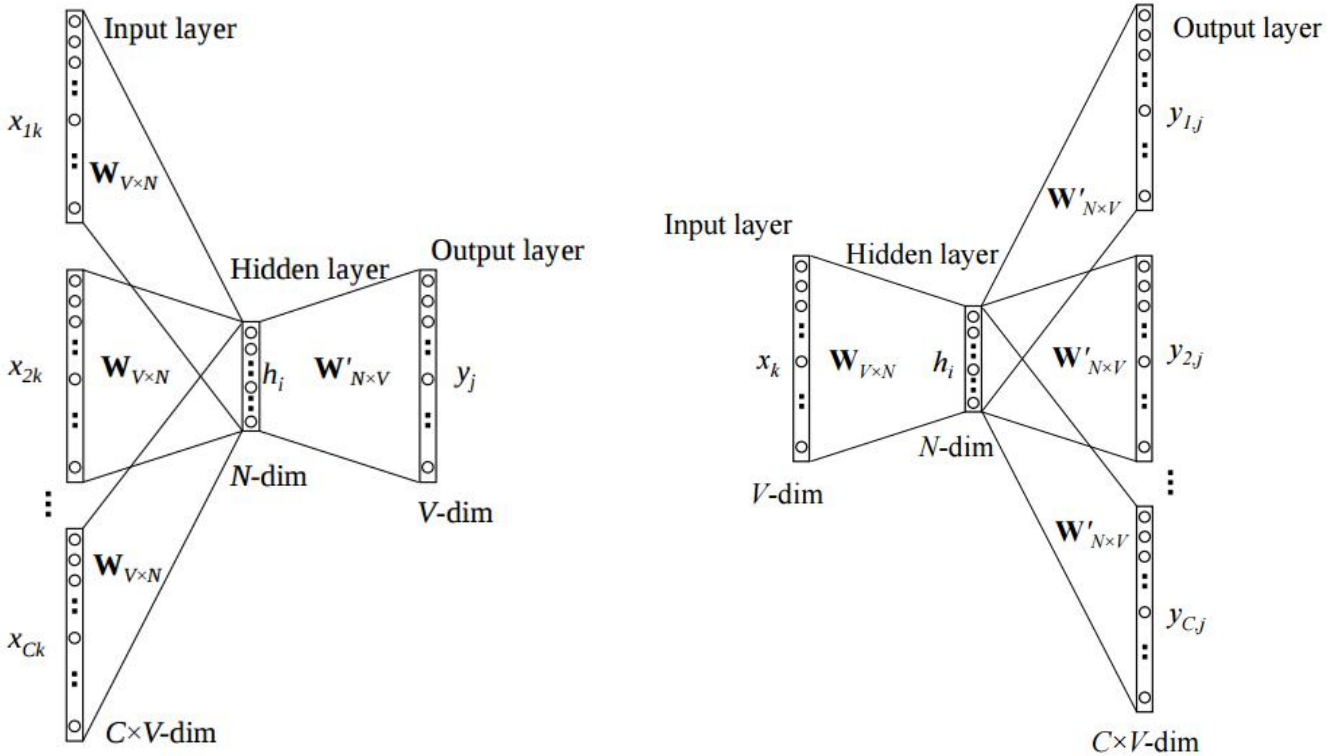
$$\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} \approx \overrightarrow{queen}$$

- GloVe (Pennington, Socher, and Manning 2014)
  - Blend Global and Local Approaches

More on Word Embeddings

- Word2Vec Algorithms
- Intuition: We shall know the meaning of a word by the company it keeps.

Figure : Source: `http://arxiv.org/pdf/1411.2738.pdf`



More on Word Embeddings

Loss Function for the Skipgram Algorithm

$$L(W, C) = \sum_{(x_i, x_j) \in U} \ell(x_j | x_i; W, C)$$

$$= - \sum_{(x_i, x_j) \in U} log \frac{e^{w_i^T c_j}}{\sum_{n=1}^{V} e^{w_i^T c_n}}$$

where U is a set of pairs constructed by the context windows in the training corpus.

In theory, we can train this through some optimization routine like the stochastic gradient descent.

More on Word Embeddings

- Last Softmax Layer over the vocabulary is expensive
  - Negative Sampling to reduce computation (Mikolov et al. 2011)
  - Instead of softmax regression over the entire set of words, perform "logistic regression" for each word in the subset of the vocabulary.
  - The subset includes the true word and randomly sampled words ("negative samples")

Efficient Loss Function for the Skipgram Algorithm

$$L'(W, C) = - \sum_{(x_i, x_j) \in U} log\sigma(w_i^T c_j) + kE_{c_N \sim P_D} log\sigma(-w_i^T c_N)$$

where k denotes the number of negative samples.

More on Word Embeddings

- Intuition of the SGD applied to the Skipgram with negative sampling
  Each word has two corresponding vectors: one denoted as w and the other denoted as c. The former is used as the word vector and the other is used as the context vector. If "cat" and "bite" occur very often within a context window, we expect that $w_{cat}^T c_{bite}$ and $w_{bite}^T c_{cat}$ are large. The gradient of the loss with respect to w_i for a particular word pair $(x_i, x_j) \in U$ is:

$$-\frac{\sigma(w_i^T c_j)(1-\sigma(w_i^T c_j))}{\sigma(w_i^T c_j)} c_j = -(1-\sigma(w_i^T c_j))c_j$$

More on Word Embeddings

Hence, the gradient descent update for a particular pair (x_i, x_j) becomes:

$$w_i^+ = w_i + \eta(1 - \sigma(w_i^T c_j))c_j$$

In other words, you drag the word vector corresponding to x_i in the direction of the context vector corresponding to x_j every time you see word x_j appears in the window of word x_i. Moreover, the degree to which the vector gets dragged depends on the confidence. The more confident you are about the wrong answer, the more you pull the vector in the direction of the context vector.

- Mechanism behind the Skipgram model is still under research
  - Levy and Goldberg 2014
  - Under the assumption that the PMI matrix is reconstructable, the optimal condition achieves factorization of the shifted PMI matrix

More on Word Embeddings

- GloVe algorithm adds "global loss" to the local loss.

$$L_{glove}(W, C) = \sum_{i,j} f(P_{ij})(w_i^T c_j - logP_{ij})^2$$

- GloVe achieves the state of the art in many extrinsic task, such as word analogies, syntactic tasks

We use the GloVe 100 dimensional vectors for our task.

Regularization

- Neural Nets often suffer over-fitting.
- Dropout deliberately corrupts the training set for generalization
- Multiplicative Noise
  $\widetilde{x}^{(i)} = x^{(i)} \odot \xi_i$ where $\xi_i \sim \frac{Bern(1-\delta)}{1-\delta}$
- Wager, Wang, and Liang proves that for logistic/softmax regression (one-layered neural networks), the dropout training performs adaptive regularization
- Dropout on recurrent neural nets and multi-layered neural nets is under research

Outline

Our Architecture

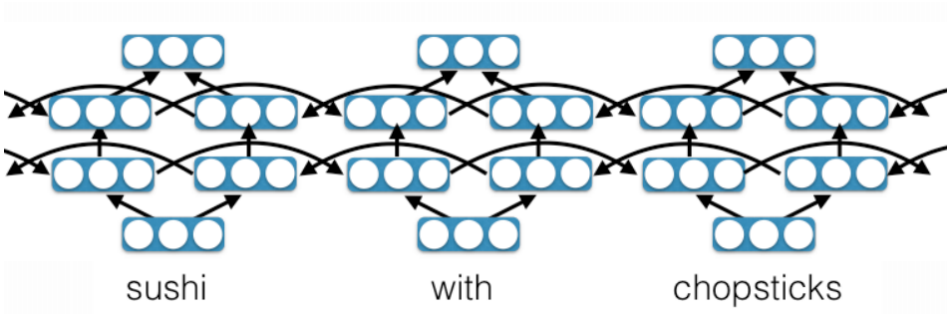


Figure : Source:
`http://www.cl.cam.ac.uk/~sc609/talks/tag16.pdf`

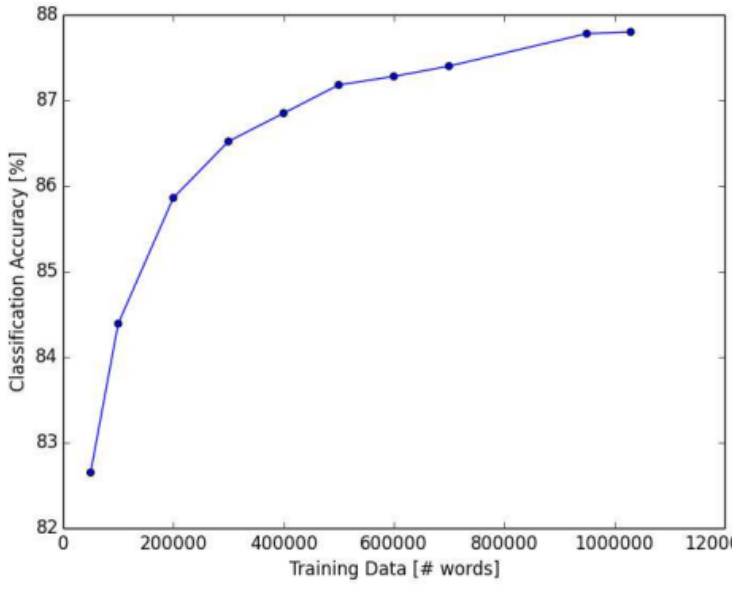
- Training Set: The Penn Tree Bank WSJ Section 1-22
- Test Set: The Penn Tree Bank WSJ Section 0
- Armed with all the building blocks above, we construct bi-directional LSTMs for supertagging.
- Run two LSTMs for both directions and concatenate the output vectors to pass them to the softmax layer
- Inputs: GloVe 100 dimensional Vectors, Predicted Part of Speech Tags (97 percent accuracy), Capitalization, Suffix

Hyperparameters and Configuration

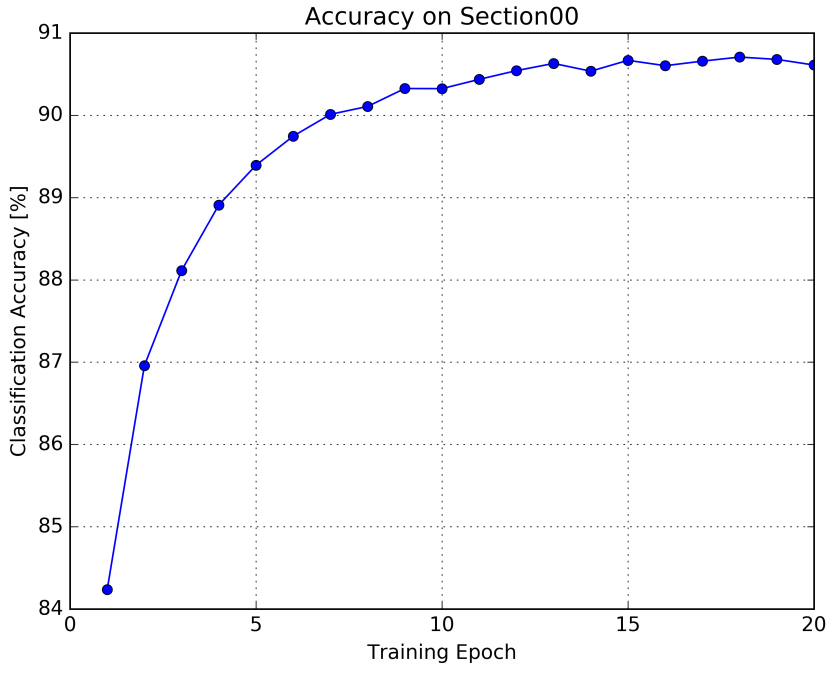
Batch Size	100
Number of Layers	2
Number of Units	512
Input Keep Probability	0.8
Hidden Units Keep Probability	0.5
Layer-to-layer Keep Probability	0.5
Optimization Algorithm	Adam with learning rate 0.001
POS tagging embedding dimensions	5

Results

(a) Chung et al. 2016



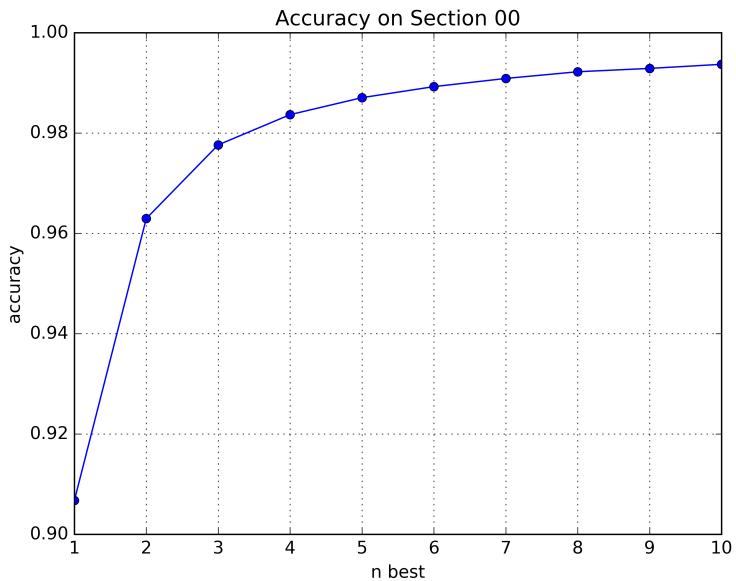
(b) Our Result



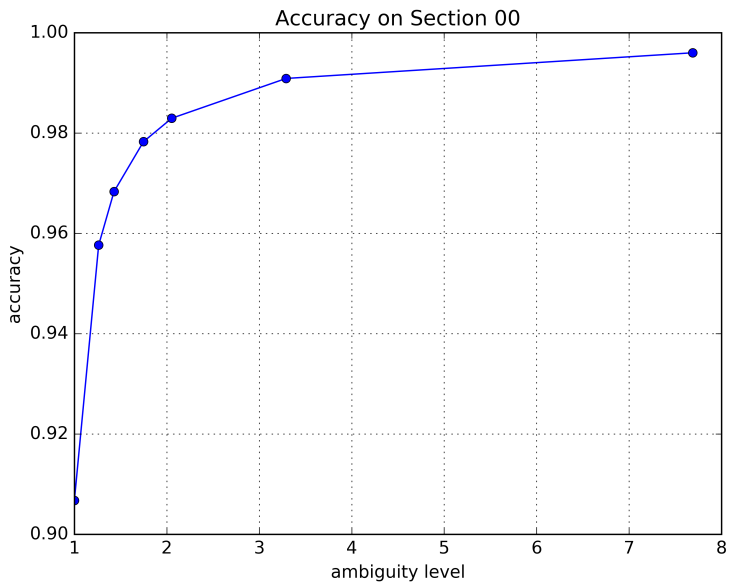
- Chung et al. use the online passive aggressive algorithm (margin-based, analogous to the Support Vector Machine)

More Results

(a) N-Best Results



(b) Beta Pruning



- $\beta = [1, 0.075, 0.03, 0.01, 0.005, 0.001, 0.0001]$
- Encourages us to take advantage of multiple candidates

Outline

Transition-based Parsing

Covered in the Final Presentation