Capturing Dependency Syntax with "Deep" Sequential Models

Yoav Goldberg
DepLing 2017
Capturing Dependency Syntax with "Deep" Sequential Models

Yoav Goldberg
DepLing 2017
Capturing Dependency Syntax with "Deep" Sequential Models

Yoav Goldberg
DepLing 2017

Eva's talk: "deep" sentential structure
Deep Learning
Deep Learning

IT LEARNS ON ITS OWN.

IT WORKS LIKE THE BRAIN.

IT CAN DO ANYTHING.
My experience with Deep Learning for Language

```
```
I'm sorry Dave,
I'm afraid I can't do that.
```

(not in the scary sense)
My experience with Deep Learning for Language

- With proper tools, easy to produce "innovative" models.
- Not so easy to get good results.
- With Feed-forward nets, hard to beat linear models w/ human engineered feature combinations.
- On 20-newsgroups, NaiveBayes+TfIdf wins over deep Feed-forward-nets and ConvNets.
My experience with Deep Learning for Language

- With proper tools, easy to produce "innovative" models.
- Not so easy to get good results.
- With Feed-forward nets, hard to beat linear models w/ human engineered feature combinations.
- On 20-newsgroups, NaiveBayes+TfIdf wins over deep Feed-forward-nets and ConvNets.
- Semi-sup learning sort-of easy with word-embeddings.
word2vec

feed in text

Text

WIKIPEDIA

wait a few hours

\[ \text{dog} = (0.12, -0.32, 0.92, 0.43, -0.3 \ldots) \]
\[ \text{cat} = (0.15, -0.29, 0.90, 0.39, -0.32 \ldots) \]
\[ \text{chair} = (0.8, 0.9, -0.76, 0.29, 0.52 \ldots) \]

get a \(|V| \times d\) matrix \(W\) where each row is a vector for a word
- dog
  - cat, dogs, dachshund, rabbit, puppy, poodle, rottweiler, mixed-breed, doberman, pig

- sheep
  - cattle, goats, cows, chickens, sheeps, hogs, donkeys, herds, shorthorn, livestock

- november
  - october, december, april, june, february, july, september, january, august, march

- jerusalem
  - tiberias, jaffa, haifa, israel, palestine, nablus, damascus katamon, ramla, safed

- teva
  - pfizer, schering-plough, novartis, astrazeneca, glaxosmithkline, sanofi-aventis, mylan, sanofi, enzyme, pharmacia
My experience with Deep Learning for Language

- With proper tools, easy to produce "innovative" models.
- Not so easy to get good results.
- With Feed-forward nets, hard to beat linear models w/ human engineered feature combinations.
- On 20-newsgroups, NaiveBayes+TfIdf wins over deep Feed-forward-nets and ConvNets.
- Semi-sup learning sort-of easy with word-embeddings.
My experience with Deep Learning for Language

• With proper tools, easy to produce "innovative" models.
• Not so easy to get good results.
• With Feed-forward nets, hard to beat linear models w/ human engineered feature combinations.
• On 20-newsgroups, NaiveBayes+TfIdf wins over deep Feed-forward-nets and ConvNets.
• Semi-sup learning sort-of easy with word-embeddings.

• **RNNs (in particular LSTMs) are really really cool.**
Doing stuff with LSTMs
Doing stuff with LSTMs
RNNS/LSTMs and Syntax
Brief intro to RNNs
Recurrent Neural Networks

- Very strong models of sequential data.
- Function from $n$ vectors to a single vector.
Recurrent Neural Networks

- Very strong models of sequential data.
- Function from $n$ vectors to a single vector.
Recurrent Neural Networks

• Very strong models of sequential data.

• Function from $n$ vectors to a single vector.
Recurrent Neural Networks

- Very strong models of sequential data.
- Function from $n$ vectors to a single vector.
Recurrent Neural Networks

• Very strong models of sequential data.

• Function from $n$ vectors to a single vector.
Recurrent Neural Networks

- Very strong models of sequential data.
- **Trainable** function from $n$ vectors to a single vector.
Recurrent Neural Networks

• There are different variants (implementations).

• We'll focus on the interface level.
Recurrent Neural Networks

- Very strong models of sequential data.
- **Trainable** function from $n$ vectors to a single vector.
Recurrent Neural Networks

\[ RNN(s_0, x_{1:n}) = s_n, y_n \]

\[ x_i \in \mathbb{R}^{d_{in}}, \ y_i \in \mathbb{R}^{d_{out}}, \ s_i \in \mathbb{R}^{f(d_{out})} \]

- Very strong models of sequential data.
- **Trainable** function from \( n \) vectors to a single* vector.
Recurrent Neural Networks

- Input vectors $\mathbf{x}_{1:i}$, output vector $\mathbf{y}_i$
- The output vector $\mathbf{y}_i$ depends on all inputs $\mathbf{x}_{1:i}$
Recurrent Neural Networks

- Recursively defined.
- There's a vector $y_i$ for every prefix $x_{1:i}$
Recurrent Neural Networks

• What are the vectors $y_i$ good for?

$$(s_0, x_1): R, O$$

• On their own? **nothing**.
Recurrent Neural Networks

- What are the vectors $y_i$ good for?

- On their own? **nothing**.

- But we can train them.
Recurrent Neural Networks

• What are the vectors $y_i$ good for?

• On their own? nothing.

• But we can train them.
Recurrent Neural Networks

- What are the vectors $y_i$ good for?

- On their own? **nothing**.

- But we can train them.
SimpleRNN:

\[ R_{SRNN}(s_{i-1}, x_i) = \tanh(W^s \cdot s_{i-1} + W^x \cdot x_i) \]

looks simple. theoretically powerful. practically, not so much.

- On their own? **nothing**.
- But we can train them.

trained parameters.

define function form
define loss
LSTM: 

\[ R_{LSTM}(s_{j-1}, x_j) = [c_j; h_j] \]

\[ c_j = c_{j-1} \odot f + g \odot i \]

\[ h_j = \text{tanh}(c_j) \odot o \]

\[ i = \sigma(W^{xi} \cdot x_j + W^{hi} \cdot h_{j-1}) \]

\[ f = \sigma(W^{xf} \cdot x_j + W^{hf} \cdot h_{j-1}) \]

\[ o = \sigma(W^{xo} \cdot x_j + W^{ho} \cdot h_{j-1}) \]

\[ g = \text{tanh}(W^{xg} \cdot x_j + W^{hg} \cdot h_{j-1}) \]

This presentation follows the recursive definition, and is correct for arbitrary long sequences. However, for a finite sized input sequence (and all input sequences we deal with are finite) one can unroll the recursion, resulting in the structure in Figure 6.

While not usually shown in the visualization, we include here the parameters \( \theta \) in order to highlight the fact that the same parameters are shared across all time steps. Different trained parameters.

• On their own? **nothing**.

• But we can train them. **define function form**

**define loss**
Recurrent Neural Networks

• What are the vectors \( y_i \) good for?

\[
\begin{align*}
R \left( s_0, x_1 \right) &= s_1, \\
y_1 &= O \left( s_1 \right), \\
R \left( s_1, x_2 \right) &= s_2, \\
y_2 &= O \left( s_2 \right), \\
R \left( s_2, x_3 \right) &= s_3, \\
y_3 &= O \left( s_3 \right), \\
R \left( s_3, x_4 \right) &= s_4, \\
y_4 &= O \left( s_4 \right), \\
R \left( s_4, x_5 \right) &= s_5, \\
y_5 &= O \left( s_5 \right)
\end{align*}
\]

The functions \( R \) and \( O \) are the same across the sequence positions, but the RNN keeps track of the states of computation through the state vector that is kept and being passed between invocations of \( R \).

Graphically, the RNN has been traditionally presented as in Figure 5.

This presentation follows the recursive definition, and is correct for arbitrary long sequences. However, for a finite sized input sequence (and all input sequences we deal with are finite) one can unroll the recursion, resulting in the structure in Figure 6.

While not usually shown in the visualization, we include here the parameters \( \theta \) in order to highlight the fact that the same parameters are shared across all time steps. Different \( \theta \) for different time steps is not usually shown.

• On their own? nothing.

• But we can train them.
Recurrent Neural Networks

Defining the loss.

**Acceptor**: predict something from end state.
Backprop the error all the way back.
Train the network to capture meaningful information.
Recurrent Neural Networks

Defining the loss.

**Acceptor:** predict something from end state.  
Backprop the error all the way back.  
Train the network to capture meaningful information
• Predict sentiment of the sentence based on all words.
• Predict word i based on words 1,...,i-1.

**Acceptor**: predict something from end state.
Backprop the error all the way back.
Train the network to capture meaningful information
• Predict sentiment of the sentence based on all words.
• **Predict word i based on words 1,...,i-1.**

**Acceptor**: predict something from end state.
Backprop the error all the way back.
Train the network to capture meaningful information.
Recurrent Neural Networks

**Transducer**: predict something from each state. Backprop the sum of errors all the way back. Train the network to capture meaningful information.
Deep RNNs

RNN can be stacked
deeper is better!
(better how?)
Story so far:

• There is a thing called a (deep) RNN.

• We can feed it a list of vectors.
  
  • Each vector represents a word.

• At the end it spits out a vector summarizing the list of vectors.

• We influence the summarization with training.
Story so far:

- There is a thing called a (deep) RNN.
- We can feed it a list of vectors.
- Each vector represents a word.
- At the end it spits out a vector summarizing the list of vectors.
- We influence the summarization with training.

**Sequence in, vector out.**

**But human language is not (only) a sequence!**

- At the end it spits out a vector summarizing the list of vectors.
- We influence the summarization with training.
Can RNNs learn hierarchy?
Can RNNs learn hierarchy?
(joint work with Tal Linzen and Emmanuel Dupoux)

Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies

Tal Linzen\textsuperscript{1,2} Emmanuel Dupoux\textsuperscript{1}
LSCP\textsuperscript{1} \& IJN\textsuperscript{2}, CNRS,
EHESS and ENS, PSL Research University

Yoav Goldberg
Computer Science Department
Bar Ilan University
yoav.goldberg@gmail.com
The case for Syntax

- Some natural-language phenomena are indicative of hierarchical structure.
- For example, subject verb agreement.

the boy kicks the ball
the boys kick the ball
The case for Syntax

- Some natural-language phenomena are indicative of hierarchical structure.

- For example, subject verb agreement.

the boy with the white shirt with the blue collar kicks the ball
the boys with the white shirts with the blue collars kick the ball
The case for Syntax

• Some natural-language phenomena are indicative of hierarchical structure.

• For example, subject verb agreement.

the boy (with the white shirt (with the blue collar)) kicks the ball

the boys (with the white shirts (with the blue collars)) kick the ball
The case for Syntax

- Some natural-language phenomena are indicative of hierarchical structure.
- For example, subject-verb agreement.

the boy (with the white shirt (with the blue collar)) kicks the ball
the boys (with the white shirts (with the blue collars)) kick the ball
Can a sequence LSTM learn agreement?
Can a sequence LSTM learn agreement?

Some prominent figures in the history of philosophy who have defended moral rationalism are Plato and Immanuel Kant.
Can a sequence LSTM learn agreement?

Some prominent figures in the history of philosophy who have defended moral are Plato and Immanuel Kant.

Replace rare words with their POS.
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral are plato and immanuel kant.

choose a verb with a subject
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral

cut the sentence at the verb
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral ___

plural or singular?

binary prediction task
... v(have) v(defended) v(moral) v(NN)
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral ... plural or singular?
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral NN ___

plural or singular?
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral NN ___

plural or singular?
Can a sequence LSTM learn agreement?

Some prominent figures in the history of philosophy who have defended moral [NN ____]

Plural or singular?
Can a sequence LSTM learn agreement?

some prominent figures in the history of philosophy who have defended moral __________

plural or singular?

in order to answer:

Need to learn the concept of number.

Need to identify the subject (ignoring irrelevant words)
Somewhat Harder Task
some prominent figures in the history of philosophy who have defended moral are plato and immanuel kant.
some prominent figures in the history of philosophy who have defended moral are plato and immanuel kant.

choose a verb with a subject and flip its number.
Some prominent figures in the history of philosophy who have defended moral are Plato and Immanuel Kant.

can the LSTM learn to distinguish good from bad sentences?
... v(boy) v(kicks) v(the) v(ball)
... v(boy)  v(kick)  v(the)  v(ball)
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well.

predicts number with 99\% accuracy.

...but most examples are very easy (look at last noun).
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well.

...but most examples are very easy (look at last noun).

![Graph showing error rate as a function of distance (no intervening nouns)]

predicts number with 99% accuracy.
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well.

- Predicts number with 99% accuracy.
- ...but most examples are very easy (look at last noun).
- When restricted to cases of at least one intervening noun:
  - 97% accuracy
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well.

Learns number of nouns
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well.

more errors as the number of intervening nouns of opposite number increases
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well. For example, a model predicts number with 99% accuracy. However, most examples are very easy when restricted to cases of at least one intervening noun: ~97% accuracy.

Figure 2: Error rates of the LSTM number prediction model as a function of: (a) distance between the subject and the verb, in dependencies that have no intervening nouns; (b) presence and number of last intervening noun; (c) count of attractors in dependencies with homogeneous intervention; (d) presence of a relative clause with and without an overt relativizer in dependencies with homogeneous intervention and exactly one attractor. All error bars represent 95% binomial confidence intervals.

We next tested whether the effect of attractors is cumulative, by focusing on dependencies with multiple attractors. To avoid cases in which the effect of an attractor is offset by an intervening noun with the same number as the subject, we restricted our search to dependencies in which all of the intervening nouns had the same number, which we term dependencies with homogeneous intervention. For example, (9) has homogeneous intervention whereas (10) does not:

(9) The roses in the vase by the door are red.
(10) The roses in the vase by the chairs are red.

Figure 2c shows that error rates increased gradually as more attractors intervened between the subject and the verb. Performance degraded quite slowly, however: even with four attractors the error rate was only 17.6%. As expected, the noun-only baselines performed significantly worse in this setting, reaching an error rate of up to 84% (worse than chance) in the case of four attractors. This confirms that syntactic cues are critical for solving the harder cases.
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well. Figure 2 shows error rates of the LSTM number prediction model as a function of:

(a) distance between the subject and the verb, in dependencies that have no intervening nouns;
(b) presence and number of last intervening noun;
(c) count of attractors in dependencies with homogeneous intervention;
(d) presence of a relative clause with and without an overt relativizer in dependencies with homogeneous intervention and exactly one attractor. All error bars represent 95% binomial confidence intervals.

Figure 2(e-f) shows additional plots:
(e) count of attractors per dependency in the corpus (note that the y-axis is on a log scale);
(f) embeddings of singular and plural nouns, projected onto their first two principal components.

Order of the sentence. We first focus on whether or not there were any intervening nouns, and if there were, whether the number of the subject differed from the number of the last intervening noun—the type of noun that would trip up the simple heuristic of agreeing with the most recent noun.

As Figure 2b shows, a last intervening noun of the same number as the subject increased error rates only moderately, from 0.4% to 0.7% in singular subjects and from 1% to 1.4% in plural subjects. On the other hand, when the last intervening noun was an agreement attractor, error rates increased by almost an order of magnitude (to 6.5% and 5.4% respectively).

Note, however, that even an error rate of 6.5% is quite impressive considering uninformed strategies such as random guessing (50% error rate), always assigning the more common class label (32% error rate, since 32% of the subjects in our corpus are plural) and the number-of-most-recent-noun heuristic (100% error rate). The noun-only LSTM baselines performed much worse in agreement attraction cases, with error rates of 46.4% (common nouns) and 40% (all nouns).

We next tested whether the effect of attractors is cumulative, by focusing on dependencies with multiple attractors. To avoid cases in which the effect of an attractor is offset by an intervening noun with the same number as the subject, we restricted our search to dependencies in which all of the intervening nouns had the same number, which we term dependencies with homogeneous intervention. For example, (9) has homogeneous intervention whereas (10) does not:

(9) The roses in the vase by the door are red.
(10) The roses in the vase by the chairs are red.

Figure 2c shows that error rates increased gradually as more attractors intervened between the subject and the verb. Performance degraded quite slowly, however: even with four attractors the error rate was only 17.6%. As expected, the noun-only baselines performed significantly worse in this setting, reaching an error rate of up to 84% (worse than chance) in the case of four attractors. This confirms that syntactic cues are critical for solving the harder cases.

but < 16% err for 4 misleading nouns...
Can a sequence LSTM learn agreement?

LSTMs learn agreement remarkably well.

but we trained it on the agreement task.

does a language model learn agreement?
Can a sequence LSTM learn agreement? does a language model learn agreement? 

Figure 4: Alternative tasks and additional experiments: (a) overall error rate across tasks (note that the y-axis ends in 10%); (b) effect of count of attractors in homogeneous dependencies across training objectives; (c) comparison of the Google LM (Jozefowicz et al., 2016) to our LM and one of our supervised verb inflection systems, on a sample of sentences; (d) number prediction: effect of count of attractors using SRNs with standard training or LSTM with targeted training; (e) number prediction: difference in error rate between singular and plural subjects across RNN cell types. Error bars represent binomial 95% confidence intervals.

Comparison to a large-scale language model: One objection to our language modeling result is that our LM faced a much harder objective than our other models—predicting a distribution over 10,000 vocabulary items is certainly harder than binary classification—but was equipped with the same capacity (50-dimensional hidden state and word vectors). Would the performance gap between the LM and the explicitly supervised models close if we increased the capacity of the LM?

We address this question using a very large publicly available LM (Jozefowicz et al., 2016), which we refer to as the Google LM.

The Google LM represents the current state-of-the-art in language modeling: it is trained on a billion-word corpus (Chelba et al., 2013), with a vocabulary of 800,000 words. It is based on a two-layer LSTM with 8192 units in each layer, or more than 300 times as many units as our LM; at 1.04 billion parameters it has almost 2 times the capacity of our LM.
Can a sequence LSTM learn agreement?

does a language model learn agreement?

what if we used the best LM in the world?
Can a sequence LSTM learn agreement?

does a language model learn agreement?

![Graph](https://github.com/tensorflow/models/tree/master/lm_1b)

Google's beast LM does better than ours but still struggles considerably.
Can a sequence LSTM learn agreement?

does a language model learn agreement?

LSTM-LM **does not** learn agreement.

Explicit error signal is required.

but with explicit signal,
LSTMs can learn agreement very well.
Can a sequence LSTM learn agreement?

Where do LSTMs fail?

in many and diverse cases.

but we did manage to find some common trends.
Can a sequence LSTM learn agreement?

Where do LSTMs fail?

noun compounds can be tricky

Conservation refugees live in a world colored in shades of gray; limbo.
Can a sequence LSTM learn agreement?

Where do LSTMs fail?

Relative clauses are hard.

The landmarks that this article lists here are also run-of-the-mill and not notable.
Can a sequence LSTM learn agreement?

Where do LSTMs fail?

Reduced relative clauses are harder.

The landmarks this article lists here are also run-of-the-mill and not notable.
Can a sequence LSTM learn agreement?

Where do LSTMs fail?

<table>
<thead>
<tr>
<th>Clause Type</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>No relative clause</td>
<td>3.2%</td>
</tr>
<tr>
<td>Overt relative clause</td>
<td>9.9%</td>
</tr>
<tr>
<td>Reduced Relative clause</td>
<td><strong>25%</strong></td>
</tr>
</tbody>
</table>
Can a sequence LSTM learn agreement?

Where do LSTMs fail?

<table>
<thead>
<tr>
<th>Clauses</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>No relative clause</td>
<td>3.2%</td>
</tr>
<tr>
<td>Overt relative clause</td>
<td>9.9%</td>
</tr>
<tr>
<td>Reduced Relative clause</td>
<td>25%</td>
</tr>
</tbody>
</table>

humans also fail much more on reduced relatives.
The agreement experiment: recap

• We wanted to show LSTMs can't learn hierarchy.
  • --> We sort-of failed.

• LSTMs learn to cope with natural-language patterns that exhibit hierarchy, based on minimal and indirect supervision.

• But some sort of relevant supervision is required.
What happens beyond English?

• English is a simple language.

• We started exploring more interesting ones.

• If you want to collaborate on cool agreement patterns in your favorite language, let's discuss!
Story so far:

• RNNs are very flexible sequence encoders.

• We can train them to encode rather intricate syntactic structures.
Story so far:

- RNNs are very flexible sequence encoders.
- We can train them to encode rather intricate syntactic structures.
- **Can we use them for parsing?**
Abstract

We suggest a compositional vector representation of parse trees that relies on a recursive combination of recurrent-neural network encoders. To demonstrate its effectiveness, we use the representation as the backbone of a greedy, bottom-up dependency parser, achieving very strong accuracies for English and Chinese, without relying on external word embeddings. The parser's implementation is available for download at the first author's webpage.

1 Introduction

Dependency-based syntactic representations of sentences are central to many language processing tasks (Kübler et al., 2009). Dependency parse-trees encode not only the syntactic structure of a sentence but also many aspects of its semantics.

A recent trend in NLP is concerned with encoding sentences as vectors (“sentence embeddings”), which can then be used for further prediction tasks. Recurrent neural networks (RNNs) (Elman, 1990), and in particular methods based on the LSTM architecture (Hochreiter and Schmidhuber, 1997), work very well for modeling sequences, and constantly obtain state-of-the-art results on both language-modeling and prediction tasks (see, e.g. (Mikolov et al., 2010)).

Several works attempt to extend recurrent neural networks to work on trees (see Section 8 for a brief overview), giving rise to the so-called recursive neural networks (Goller and Kuchler, 1996; Socher et al., 2010). However, recursive neural networks do not cope well with trees with arbitrary branching factors – most work require the encoded trees to be binary-branching, or have a fixed maximum arity. Other attempts allow arbitrary branching factors, at the expense of ignoring the order of the modifiers. In contrast, we propose a tree-encoding that naturally supports trees with arbitrary branching factors, making it particularly appealing for dependency trees. Our tree encoder uses recurrent neural networks as a building block: we model the left and right sequences of modifiers using RNNs, which are composed in a recursive manner to form a tree (Section 3). We use our tree representation for encoding the partially-built parse trees in a greedy, bottom-up dependency parser which is based on the easy-first transition-system of Goldberg and Elhadad (2010).

Using the Hierarchical Tree LSTM representation, and without using any external embeddings, our parser achieves parsing accuracies of 92.6 UAS and 90.2 LAS on the PTB (Stanford dependencies) and 86.1 UAS and 84.4 LAS on the Chinese treebank, while relying on greedy decoding.

To the best of our knowledge, this is the first work to demonstrate competitive parsing accuracies for full-scale parsing while relying solely on recursive, compositional tree representations, and without using a reranking framework. We discuss related work in Section 8.

While the parsing experiments demonstrate the suitability of our representation for capturing the structural elements in the parse tree that are useful for predicting parsing decisions, we are interested in exploring the use of the RNN-based compositional vector representation of parse trees also for semantic tasks.
Core Idea

• LSTMs are SOTA at modeling sequences.
• Encode sequence of modifiers as an LSTM.
• Combine in a recursive manner.

▶ great for dependency trees.
Core Idea

- **Two LSTMs**
  - head + Left modifiers encoded w/ LSTM-L
  - head + Right modifiers encoded w/ LSTM-R
  - The Left and Right end states are concatenated
Core Idea

- Two LSTMs
- **head + Left modifiers encoded w/ LSTM-L**
- head + Right modifiers encoded w/ LSTM-R
- The Left and Right end states are concatenated
- Two LSTMs
- head + Left modifiers encoded w/ LSTM-L
- head + Right modifiers encoded w/ LSTM-R
- The Left and Right end states are concatenated
Core Idea

- Two LSTMs
- head + Left modifiers encoded w/ LSTM-L
- head + Right modifiers encoded w/ LSTM-R
- The Left and Right end states are concatenated
the black fox who really likes apples did not jump over a lazy dog yesterday
the black fox who really likes apples did not jump over a lazy dog yesterday
the black fox who really likes apples did not jump over a lazy dog yesterday
the black fox who really likes apples did not jump over a lazy dog yesterday
the black fox who really likes apples did not jump over a lazy dog yesterday
Hierarchical Tree LSTM (Example)

```
the  brown  fox  jumped  with  joy  over  the  fence
```

12
Hierarchical Tree LSTM (Example)

```
Hierarchical Tree LSTM (Example)

the brown fox jumped with joy over the fence

Sub Tree (fox)

LeftChildren(LSTM)  RightChildren(LSTM)
the  brown  fox     fox
```
Hierarchical Tree LSTM (Example)

the brown fox jumped with joy over the fence

Sub Tree (fox)
LeftChildren(LSTM) RightChildren(LSTM)
the brown fox

Sub Tree (with)
LeftChildren(LSTM) RightChildren(LSTM)
with joy
Hierarchical Tree LSTM (Example)

the brown fox jumped with joy over the fence

Sub Tree (fox)

Sub Tree (with)

Sub Tree (over)
Hierarchical Tree LSTM (Example)

the brown fox jumped with joy over the fence

Sub Tree (jumped)

LeftChildren(LSTM) RightChildren(LSTM)

the brown fox jumped jumped with joy over the fence

Sub Tree (fox)

LeftChildren(LSTM) RightChildren(LSTM)

the brown fox fox

Sub Tree (with)

LeftChildren(LSTM) RightChildren(LSTM)

with with joy

Sub Tree (over)

LeftChildren(LSTM) RightChildren(LSTM)

over over the fence
Hierarchical Tree LSTM (Example)

the brown fox jumped with joy over the fence

Sub Tree (jumped)

LeftChildren(LSTM)

the brown fox

RightChildren(LSTM)

jumped

with joy

over the fence

Sub Tree (fox)

LeftChildren(LSTM)

the brown

RightChildren(LSTM)

fox

Sub Tree (with)

LeftChildren(LSTM)

with

RightChildren(LSTM)

with joy

Sub Tree (over)

LeftChildren(LSTM)

over

RightChildren(LSTM)

over the fence
Hierarchical Tree LSTM (Example)

- the brown fox jumped with joy over the fence

Sub Tree (jumped)

LeftChildren(LSTM)
- the brown fox jumped

RightChildren(LSTM)
- jumped with joy over the fence

Sub Tree (fox)

LeftChildren(LSTM)
- the brown fox

RightChildren(LSTM)
- fox

Sub Tree (with)

LeftChildren(LSTM)
- with

RightChildren(LSTM)
- with joy

Sub Tree (over)

LeftChildren(LSTM)
- over

RightChildren(LSTM)
- over the fence
Hierarchical Tree LSTM (Example)

the brown fox jumped with joy over the fence

Sub Tree (jumped)

LeftChildren (LSTM)
the brown fox jumped

RightChildren (LSTM)
jumped with joy over the fence

Sub Tree (fox)

LeftChildren (LSTM)
the brown fox

RightChildren (LSTM)
fox

Sub Tree (with)

LeftChildren (LSTM)
with

RightChildren (LSTM)
with joy

Sub Tree (over)

LeftChildren (LSTM)
over

RightChildren (LSTM)
over the fence
How Do We Capture Leafs?

the brown fox jumped with joy over the fence

Sub Tree (jumped)

LeftChildren(LSTM)
the brown fox jumped

RightChildren(LSTM)
jumped with joy over the fence

Sub Tree (fox)

LeftChildren(LSTM)
the brown fox

RightChildren(LSTM)
fox

Sub Tree (with)

LeftChildren(LSTM)
with

RightChildren(LSTM)
joy

Sub Tree (over)

LeftChildren(LSTM)
over

RightChildren(LSTM)
over the fence
Easy-First Parsing (Score Function)

How do we efficiently compute the Hierarchical Tree LSTM representation for each pending sub-tree?
How do we efficiently compute the Hierarchical Tree LSTM representation for each pending sub-tree?
Easy-First Parsing (Score Function)

How do we efficiently compute the Hierarchical Tree LSTM representation for each pending sub-tree?
Easy-First Parsing (Score Function)

How do we efficiently compute the Hierarchical Tree LSTM representation for each pending sub-tree?
Bottom-Up Tree Representation Building
Bottom-Up Tree Representation Building

```
the
  \rightarrow
fox
  \rightarrow jumped
  \rightarrow with
  \rightarrow joy

brown
  \rightarrow
```

Sub Tree (fox)

Sub Tree (fox)

LeftChildren (LSTM)

RightChildren (LSTM)

19
Bottom-Up Tree Representation Building

the brown fox jumped with joy

Sub Tree (fox)

19
Bottom-Up Tree Representation Building

the brown fox jumped with joy

↓

fox jumped with joy

the brown
Bottom-Up Tree Representation Building

The brown fox jumped with joy.

Sub Tree (fox)

Left Children (LSTM)
- brown
- fox

Right Children (LSTM)
- fox
Bottom-Up Tree Representation Building

the brown fox jumped with joy

LeftChildren (LSTM)  RightChildren (LSTM)

brown  fox  fox

Sub Tree (fox)
Bottom-Up Tree Representation Building

Constant Time!
Easy-First Parsing (Continued)
Easy First Parsing with Hierarchical Tree LSTMs

- It works! We get nice results.

- But.

- Turns out can actually do something much simpler.
Abstract
We present a simple and effective scheme for dependency parsing which is based on bidirectional-LSTMs (BiLSTMs). Each sentence token is associated with a BiLSTM vector representing the token in its sentential context, and feature vectors are constructed by concatenating a few BiLSTM vectors. The BiLSTM is trained jointly with the parser objective, resulting in very effective feature extractors for parsing. We demonstrate the effectiveness of the approach by applying it to a greedy transition based parser as well as to a globally optimized graph-based parser. The resulting parsers have very simple architectures, and match or surpass the state-of-the-art accuracies on English and Chinese.

1 Introduction
The focus of this paper is on feature representation for dependency parsing, using recent techniques from the neural-networks (“deep learning”) literature. Modern approaches to dependency parsing can be broadly categorized into graph-based and transition-based parsers (Kübler et al., 2008). Graph-based parsers (McDonald, 2006) treat parsing as a search-based structured prediction problem in which the goal is learning a scoring function over dependency trees such that the correct tree is scored above all other trees. Transition-based parsers (Nivre, 2004; Nivre, 2008) treat parsing as a sequence of actions that produce a parse tree, and a classifier is trained to score the possible actions at each stage of the process and guide the parsing process. Perhaps the simplest graph-based parsers are arc-factored (first order) models (McDonald, 2006), in which the scoring function for a tree decomposes over the individual arcs of the tree. More elaborate models look at larger (overlapping) parts, requiring more sophisticated inference and training algorithms (Martins et al., 2009; Koo and Collins, 2010). The basic transition-based parsers work in a greedy manner, performing a series of locally-optimal decisions, and boast very fast parsing speeds. More advanced transition-based parsers introduce some search into the process using a beam (Zhang and Clark, 2008) or dynamic programming (Huang and Sagae, 2010).

Regardless of the details of the parsing framework being used, a crucial step in parser design is choosing the right feature function for the underlying statistical model. Recent work (see Section 2.2 for an overview) attempt to alleviate parts of the feature function design problem by moving from linear to non-linear models, enabling the modeler to focus on a small set of “core” features and leaving it up to the machine-learning machinery to come up with good feature combinations (Chen and Manning, 2014; Pei et al., 2015; Lei et al., 2014; Taub-Tabib et al., 2015). However, the need to carefully define a set of core features remains. For example, the work of (Chen and Manning, 2014) uses 18 different elements in its feature function, while the work of (Pei et al., 2015) uses 21 different elements. Other works, notably (Dyer et al., 2015; Le and Zuidema, 2014), propose more sophisticated feature representations, in which the feature engineering is replaced with architecture engineering.

In this work, we suggest an approach which is much simpler in terms of both feature engineering
Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations

Eliyahu Kiperwasser
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
elikip@gmail.com

Yoav Goldberg
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
yoav.goldberg@gmail.com

Abstract
We present a simple and effective scheme for dependency parsing which is based on bidirectional-LSTMs (BiLSTMs). Each sentence token is associated with a BiLSTM vector representing the token in its sentential context, and feature vectors are constructed by concatenating a few BiLSTM vectors. The BiLSTM is trained jointly with the parser objective, resulting in very effective feature extractors for parsing. We demonstrate the effectiveness of the approach by applying it to a greedy transition based parser as well as to a globally optimized graph-based parser. The resulting parsers have very simple architectures, and match or surpass the state-of-the-art accuracies on English and Chinese.

1 Introduction
The focus of this paper is on feature representation for dependency parsing, using recent techniques from the neural-networks (“deep learning”) literature. Modern approaches to dependency parsing can be broadly categorized into graph-based and transition-based parsers (Kübler et al., 2008). Graph-based parsers (McDonald, 2006) treat parsing as a search-based structured prediction problem in which the goal is learning a scoring function over dependency trees such that the correct tree is scored above all other trees. Transition-based parsers (Nivre, 2004; Nivre, 2008) treat parsing as a sequence of actions that produce a parse tree, and a classifier is trained to score the possible actions at each stage of the process and guide the parsing process. Perhaps the simplest graph-based parsers are arc-factored (first order) models (McDonald, 2006), in which the scoring function for a tree decomposes over the individual arcs of the tree. More elaborate models look at larger (overlapping) parts, requiring more sophisticated inference and training algorithms (Martins et al., 2009; Koo and Collins, 2010). The basic transition-based parsers work in a greedy manner, performing a series of locally-optimal decisions, and boast very fast parsing speeds. More advanced transition-based parsers introduce some search into the process using a beam (Zhang and Clark, 2008) or dynamic programming (Huang and Sagae, 2010).

Regardless of the details of the parsing framework being used, a crucial step in parser design is choosing the right feature function for the underlying statistical model. Recent work (see Section 2.2 for an overview) attempt to alleviate parts of the feature function design problem by moving from linear to non-linear models, enabling the modeler to focus on a small set of “core” features and leaving it up to the machine-learning machinery to come up with good feature combinations (Chen and Manning, 2014; Pei et al., 2015; Lei et al., 2014; Taub-Tabib et al., 2015). However, the need to carefully define a set of core features remains. For example, the work of (Chen and Manning, 2014) uses 18 different elements in its feature function, while the work of (Pei et al., 2015) uses 21 different elements. Other works, notably (Dyer et al., 2015; Le and Zuidema, 2014), propose more sophisticated feature representations, in which the feature engineering is replaced with architecture engineering.

In this work, we suggest an approach which is much simpler in terms of both feature engineering
Bi-directional RNNs
RNNs so far:

Each state encodes the entire history up to that state. This is not bad. **But what about the future?**
Bidirectional RNNs

One RNN runs left to right.
Another runs right to left.
Encode both future and history of a word.
Bidirectional RNNs

One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.
Bidirectional RNNs

One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.
BI-RNNs

One RNN runs left to right. Another runs right to left. Encode both future and history of a word.
BI-RNNs

One RNN runs left to right. Another runs right to left. Encode *both future and history* of a word.

$$\text{BiRNN}(x_{1:7}, 4) = [y^F_4; y^R_4]$$

$$y^F_4 = RNN_F(x_{1:4})$$

$$y^R_4 = RNN_R(x_{7:4})$$
Deep BI-RNNs

BI-RNN can also be stacked
(Deep) BI-RNNs

• provide an "infinite" window around a focus word.
• learn to extract what's important.
• easy to train!
• very effective for sequence tagging.
• Great as feature extractors!
Simple and Accurate Dependency Parsing
Using Bidirectional LSTM Feature Representations

Eliyahu Kiperwasser
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
elikip@gmail.com

Yoav Goldberg
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
yoav.goldberg@gmail.com
There are two main frameworks for parsing:

- **Graph-based:**
  - Global inference
  - Score factorized over parts
  - There are first, second & third order parsers.

- **Transition-based:**
  - Greedy local inference
  - Score relies on current configuration, which is dependent on all previous transitions
There are two main frameworks for parsing:

- **Graph-based:**
  - Global inference
  - Score factorized over parts
  - There are first, second & third order parsers.

- **Transition-based:**
  - Greedy local inference
  - Score relies on current configuration, which is dependent on all previous transitions
Structured Prediction Recipe

\[
predict(x) = \arg \max_{y \in Y(x)} \sum_{p \in y} \text{score}(\phi(p))
\]

- Decompose structure to local factors.
- Assign a score to each factor.
- Structure score = sum of local scores.
- Look for highest scoring structure.
Graph-based Parsing

Score(They ate the pizza with anchovies) =

Score(They ate) + Score(ate pizza) + Score(the pizza) + Score(pizza with) + Score(with anchovies)
Graph-based Parsing (Inference)

Input Sentence: "They ate pizza"

Score(ate → pizza)
Score(they → ate)
Score(pizza → ate)
Score(pizza → they)
Score(they → pizza)
Score(root → pizza)
Score(root → ate)
Score(root → they)

Eliyahu Kiperwasser (Bar-Ilan University)
Graph-based Parsing (Inference)

Spanning tree with maximal score
Structured Prediction Recipe

\[ predict(x) = \arg \max_{y \in \mathcal{Y}(x)} \sum_{p \in y} \text{score}(\phi(p)) \]

- *feature function* extracts useful signals from parts.
- most work goes into this component.
Arc Score Function

\[ \text{Score}( \text{modifier} \rightarrow \text{head} ) = ? \]
Arc Score Function

\[
\text{Score( modifier, head ) } = F( \phi(\text{modifier, head; sentence}) )
\]

- Similar story for transition-based parser
- The choice of features is very important
First-order features
(from Ryan McDonald's PhD thesis)

- Words and POS of Head and Mod.
- Words and POS of neighbors of Head and Mod.
- POS between Head and Modifier.
- Distance between Head and Modifier.
- Direction between Head and Modifier.
- Many, many combination features.
First-order features
(from Ryan McDonald's PhD thesis)

<table>
<thead>
<tr>
<th>Basic Uni-gram Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$-word, $x_i$-pos</td>
</tr>
<tr>
<td>$x_i$-word</td>
</tr>
<tr>
<td>$x_i$-pos</td>
</tr>
<tr>
<td>$x_j$-word, $x_j$-pos</td>
</tr>
<tr>
<td>$x_j$-word</td>
</tr>
<tr>
<td>$x_j$-pos</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Bi-gram Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$-word, $x_i$-pos, $x_j$-word, $x_j$-pos</td>
</tr>
<tr>
<td>$x_i$-pos, $x_j$-word, $x_j$-pos</td>
</tr>
<tr>
<td>$x_i$-word, $x_j$-word, $x_j$-pos</td>
</tr>
<tr>
<td>$x_i$-word, $x_i$-pos, $x_j$-pos</td>
</tr>
<tr>
<td>$x_i$-word, $x_j$-pos, $x_j$-word</td>
</tr>
<tr>
<td>$x_i$-pos, $x_j$-pos</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In Between POS Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$-pos, b-pos, $x_j$-pos</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Surrounding Word POS Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$-pos, $x_i$-pos+1, $x_j$-pos-1, $x_j$-pos</td>
</tr>
<tr>
<td>$x_i$-pos-1, $x_i$-pos, $x_j$-pos-1, $x_j$-pos</td>
</tr>
<tr>
<td>$x_i$-pos, $x_i$-pos+1, $x_j$-pos, $x_j$-pos+1</td>
</tr>
<tr>
<td>$x_i$-pos-1, $x_i$-pos, $x_j$-pos, $x_j$-pos+1</td>
</tr>
</tbody>
</table>

Table 3.1: Features used by system, $f(i,j)$, where $x_i$ is the head and $x_j$ the modifier in the dependency relation. $x_i$-word: word of head in dependency edge. $x_j$-word: word of modifier. $x_i$-pos: POS of head. $x_j$-pos: POS of modifier. $x_i$-pos+1: POS to the right of head in sentence. $x_i$-pos-1: POS to the left of head. $x_j$-pos+1: POS to the right of modifier. $x_j$-pos-1: POS to the left of modifier. b-pos: POS of a word in between head and modifier.
Core Features + Feature Combinations

Example from slides of Rush and Petrov (2012)
Core Features + Feature Combinations

replace feature combinations with non-linear learner

Figure from Chen and Manning (2014)
Similar approach in Pei et al, Weiss et al, Andor et al
replace feature combinations with non-linear learner

but still need to define good features.

Figure from Chen and Manning (2014)
Similiar approach in Pei et al, Weiss et al, Andor et al
Our take on it

Let's just use a Bidirectional LSTM
The fox who likes apples jumped over a dog.

\[ \phi(x, \text{jumped}, \text{fox}) \]

\[ \text{score}(h, m, x) = MLP(\phi(x, h, m)) \]

\[ \phi(x, h, m) = [BIRNN(x, h); BIRNN(x, m)] \]
$score(h, m, x) = MLP(\phi(x, h, m))$

$\phi(x, h, m) = [BIRNN(x, h); BIRNN(x, m)]$
\[ \text{score}(h, m, x) = MLP(\phi(x, h, m)) \]
\[ \phi(x, h, m) = [BIRNN(x, h); BIRNN(x, m)] \]
score(h, m, x) = MLP(ϕ(x, h, m))

ϕ(x, h, m) = [BIRNN(x, h); BIRNN(x, m)]
\[ \text{score}(h, m, x) = MLP(\phi(x, h, m)) \]

\[ \phi(x, h, m) = [\text{BIRNN}(x, h); \text{BIRNN}(x, m)] \]
\[
\text{score}(h, m, x) = MLP(\phi(x, h, m)) \\
\phi(x, h, m) = [\text{BIRNN}(x, h); \text{BIRNN}(x, m)]
\]
the two BI-RNN vectors give us:

\[
\text{score}(h, m, x) = MLP(\phi(x, h, m))
\]

\[
\phi(x, h, m) = [BIRNN(x, h); BIRNN(x, m)]
\]
The BiLSTM encoding of a word holds information about its attachment preferences.

The score is dependent on the BiLSTM encoding which in turn depends on the entire sentence.

Therefore, the score function focused on a specific arc is considering also the entire sentence attachment preferences.
Tree Score

\[
\text{Score}(\text{They ate the pizza with anchovies}) =
\]

\[
\begin{align*}
\text{MLP} & \quad \text{MLP} \\
\text{V}_\text{they} & \quad \text{V}_\text{ate} \\
\text{concat} & \quad \text{concat} \\
\text{LSTM}_b & \quad \text{LSTM}_b \\
\text{x}_\text{they} & \quad \text{x}_\text{ate} \\
\text{MLP} & \quad \text{MLP} \\
\text{V}_\text{the} & \quad \text{V}_\text{pizza} \\
\text{concat} & \quad \text{concat} \\
\text{LSTM}_b & \quad \text{LSTM}_b \\
\text{x}_\text{the} & \quad \text{x}_\text{pizza} \\
\text{MLP} & \quad \text{MLP} \\
\text{V}_\text{with} & \quad \text{V}_\text{anchovies} \\
\text{concat} & \quad \text{concat} \\
\text{LSTM}_b & \quad \text{LSTM}_b \\
\text{x}_\text{with} & \quad \text{x}_\text{anchovies} \\
\text{MLP} & \quad \text{MLP} \\
\text{LSTM}_b & \quad \text{LSTM}_b \\
\text{x}_\text{concat} \\
\text{MLP} & \quad \text{MLP} \\
\text{LSTM}_b & \quad \text{LSTM}_b \\
\text{x}_\text{concat} \\
\text{MLP} & \quad \text{MLP} \\
\text{LSTM}_b & \quad \text{LSTM}_b \\
\text{x}_\text{concat} \\
\end{align*}
\]
Large Margin Objective

$$\max(0, 1 - \text{blue} + \text{red})$$
Training Objective

Gold tree should score a margin above all other trees

\[ \sum_{(h,m) \in y} MLP(\phi(x, h, m)) - \sum_{(h,m) \in y' \neq y} MLP(\phi(x, h, m)) > 1 \]

\[ \phi(x, h, m) = [BIRNN(x, h); BIRNN(x, m)] \]

Backdrop all the way back through the BI-LSTM
Cost Augmentation: Make non-gold attachments more attractive in training by adding a constant to their score

Multi-Task Learning: Learning the label on the same BiLSTM representation helps both in terms of accuracy and performance.

For Speed: Simple algebraic “trick” reduces the number of matrix multiplication significantly.
Graph-based Parsing (More Details)

- **Cost Augmentation**: Make non-gold attachments more attractive in training by adding a constant to their score.

- **Multi-Task Learning**: Learning the label on the same BiLSTM representation helps both in terms of accuracy and performance.

- **For Speed**: Simple algebraic “trick” reduces the number of matrix multiplication significantly.
Arc Labels (Multi-Task Learning)

- The arc labels hold important additional syntactic information.
- The labels contribute information useful for the unlabeled case too.
Arc Labels (Multi-Task Learning)

Different MLP

Same BiLSTM

Enrich BiLSTM representation by learning labels
Arc Labels (Multi-Task Learning)

Different MLP

Same BiLSTM

Enrich BiLSTM representation by learning labels
In parsing time

• Run (deep) BI-LSTM over words+POS.
  • this gives us a vector $v_i$ for each word.
• Compute scores for each arc (h,m) via $MLP([v_h; v_m])$
• Decode using arc scores.
Graph-based Parsing

and this works:

93.2 UAS with two features,
first-order parser,
without external embeddings.
and this works:

93.2 UAS with two features,

first-order parser,

without external embeddings.
This is remarkably effective!
We can use same trick also for Transition based parsing
Transition-based Parsing (Oracle)

Configuration:

\[ s_2 \quad s_1 \quad s_0 \quad b_0 \quad b_1 \quad b_2 \quad b_3 \]

\[ \text{the} \quad \text{jumped} \quad \text{over} \quad \text{the} \quad \text{the} \quad \text{lazy} \quad \text{dog} \quad \text{ROOT} \]

Scoring:

\[(\text{Score}_{\text{LeftArc}}, \text{Score}_{\text{RightArc}}, \text{Score}_{\text{Shift}})\]

Eliyahu Kiperwasser (Bar-Ilan University)  Simple and Accurate Dependency Parsing
also worth noting:

Incremental Parsing with Minimal Features Using Bi-Directional LSTM

James Cross and Liang Huang
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, Oregon, USA
{crossj, liang.huang}@oregonstate.edu

Constituency Parsing
Transition-based
also worth noting:

Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set

Tianze Shi
Cornell University
tianze@cs.cornell.edu

Liang Huang
Oregon State University
liang.huang.sh@gmail.com

Lillian Lee
Cornell University
llee@cs.cornell.edu

Dependency Parsing
Transition-based + Dynamic Programming
also worth noting:

Transition-Based Dependency Parsing with Stack Long Short-Term Memory

Chris Dyer\*\*  Miguel Ballesteros\*\*  Wang Ling\*  Austin Matthews\*  Noah A. Smith\*  
\*Marianas Labs  \*\*NLP Group, Pompeu Fabra University  \*\*Carnegie Mellon University

chris@marianaslabs.com, miguel.ballesteros@upf.edu,
{lingwang,austinma,nasmith}@cs.cmu.edu

in retrospect
"Stack LSTM" parser is very similar to the biLSTM
(but does have extra compositionality)
But let's get back to the 1st-order Graph Parser
1st order Decomposition is Incredibly Naive

\[
\text{Score}\left( \text{They ate the pizza with anchovies} \right) =
\]

\[
\text{Score}\left( \text{They ate} \right) + \text{Score}\left( \text{ate pizza} \right) + \text{Score}\left( \text{the pizza} \right) +
\]

\[
\text{Score}\left( \text{pizza with} \right) + \text{Score}\left( \text{with anchovies} \right)
\]
And yet...

RBG Parser (Lei et al, 2014), 1st order: 91.7 UAS
TurboPasrer (Martins et al, 2013), 3rd order: 93.1 UAS
BiLSTM (K&G, 2016), 1st order: 93.2 UAS
BiLSTM (K&G, 2016), + embeddings: 92.7 UAS
BiLSTM (K&G, 2016), + emb, bug fix: 94.0 UAS
Dozat and Manning 2017:
Deep Biaffine Attention for Neural Dependency Parsing

Timothy Dozat
Stanford University
tdozat@stanford.edu

Christopher D. Manning
Stanford University
manning@stanford.edu
MLP([birnn(h); birnn(m)])

K&G 2016
$MLP([birnn(h); birnn(m)])$

$\phi(x, jumped, fox)$

Dozat and Manning, 2017
the/D fox/N who/P likes/V apples/N jumped/V over/P a/D dog/N

\[ MLP([\text{birnn}(h); \text{birnn}(m)]) \]
\[ \text{birnn}(h) \text{ M birnn}(m) \]

Dozat and Manning, 2017
The fox who likes apples jumped over a dog.

$ MLP([\text{birnn}(h); \text{birnn}(m)])$

$\phi(x, \text{jumped, fox})$

Notable hyperparameters

- Lots of dropout (including embedding dropout, same-mask recurrent dropout (Gal and Ghahramani, 2016))
- Small changes to default Adam (Kingma and Ba, 2015) ($\beta_2 = .9$, only update $m_t, v_t$ for words used in the minibatch)
- Preference for zero-initializations (especially frequent-word embeddings)

Results

Dozat and Manning, 2017
And yet...

RBG Parser (Lei et al, 2014), 1st order: 91.7 UAS
TurboParser (Martins et al, 2013), 3rd order: 93.1 UAS
BiLSTM (K&G, 2016), 1st order: 93.2 UAS
BiLSTM (K&G, 2016), + embeddings: 92.7 UAS
BiLSTM (K&G, 2016), + emb, bug fix: 94.0 UAS
**Dozat and Manning 2017:** 95.7 UAS
(BiLSTM. First Order)
CoNLL 2017 Shared Task
## CoNLL 2017 Shared Task

### Results: Unlabeled Attachment Score (UAS)

#### All treebanks

<table>
<thead>
<tr>
<th>Rank</th>
<th>Team Name</th>
<th>Software</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Stanford (Stanford)</td>
<td>software1</td>
<td>81.30</td>
</tr>
<tr>
<td>2.</td>
<td>C2L2 (Ithaca)</td>
<td>software5</td>
<td>80.35</td>
</tr>
<tr>
<td>3.</td>
<td>IMS (Stuttgart)</td>
<td>software2</td>
<td>79.90</td>
</tr>
<tr>
<td>4.</td>
<td>HIT-SCIR (Harbin)</td>
<td>software4</td>
<td>77.81</td>
</tr>
<tr>
<td>5.</td>
<td>LATTICE (Paris)</td>
<td>software7</td>
<td>76.75</td>
</tr>
<tr>
<td>6.</td>
<td>NAIST SATO (Nara)</td>
<td>software1</td>
<td>76.35</td>
</tr>
<tr>
<td>7.</td>
<td>UParse (Edinburgh)</td>
<td>software1</td>
<td>75.49</td>
</tr>
<tr>
<td>8.</td>
<td>Koç University (İstanbul)</td>
<td>software3</td>
<td>75.44</td>
</tr>
<tr>
<td>9.</td>
<td>ÚFAL – UDPipe 1.2 (Praha)</td>
<td>software1</td>
<td>75.39</td>
</tr>
<tr>
<td>10.</td>
<td>Orange – Deskiň (Lannion)</td>
<td>software1</td>
<td>75.11</td>
</tr>
<tr>
<td>11.</td>
<td>RACAI (Bucureşti)</td>
<td>software1</td>
<td>74.67</td>
</tr>
<tr>
<td>12.</td>
<td>LyS-FASTPARSE (A Coruña)</td>
<td>software5</td>
<td>74.42</td>
</tr>
</tbody>
</table>
CoNLL 2017 Shared Task

Results: Unlabeled Attachment Score (UAS)

<table>
<thead>
<tr>
<th>All treebanks</th>
<th>Software 1</th>
<th>Software 2</th>
<th>Software 3</th>
<th>Software 4</th>
<th>Software 5</th>
<th>Software 6</th>
<th>Software 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Stanford (Stanford)</td>
<td>81.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. C2L2 (Ithaca)</td>
<td></td>
<td>80.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. IMS (Stuttgart)</td>
<td></td>
<td></td>
<td>79.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. HIT-SCIR (Harbin)</td>
<td></td>
<td></td>
<td></td>
<td>77.81</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. LATTICE (Paris)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. NAIST SATO (Nara)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76.35</td>
<td></td>
</tr>
<tr>
<td>7. UParse (Edinburgh)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75.49</td>
<td></td>
</tr>
<tr>
<td>8. Koç University (İstanbul)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75.44</td>
</tr>
<tr>
<td>9. ÚFAL – UDPipe 1.2 (Praha)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75.39</td>
<td></td>
</tr>
<tr>
<td>10. Orange – Deskiñ (Lannion)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75.11</td>
</tr>
<tr>
<td>11. RACAI (București)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74.67</td>
</tr>
<tr>
<td>12. LyS-FASTPARSE (A Coruña)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74.42</td>
</tr>
</tbody>
</table>

**Dozat and Manning**

biLSTM + graph + tuning (first-order features)
CoNLL 2017 Shared Task

Results: Unlabeled Attachment Score (UAS)

All treebanks

<table>
<thead>
<tr>
<th>Rank</th>
<th>Treebank</th>
<th>UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Stanford (Stanford)</td>
<td>81.30</td>
</tr>
<tr>
<td>2.</td>
<td>C2L2 (Ithaca)</td>
<td>80.35</td>
</tr>
<tr>
<td>3.</td>
<td>IMS (Stuttgart)</td>
<td>79.90</td>
</tr>
<tr>
<td>4.</td>
<td>HIT-SCIR (Harbin)</td>
<td>77.81</td>
</tr>
<tr>
<td>5.</td>
<td>LATTICE (Paris)</td>
<td>76.75</td>
</tr>
<tr>
<td>6.</td>
<td>NAIST SATO (Nara)</td>
<td>76.35</td>
</tr>
<tr>
<td>7.</td>
<td>UParse (Edinburgh)</td>
<td>75.49</td>
</tr>
<tr>
<td>8.</td>
<td>Koç University (İstanbul)</td>
<td>75.44</td>
</tr>
<tr>
<td>9.</td>
<td>ÚFAL - UDPipe 1.2 (Praha)</td>
<td>75.39</td>
</tr>
<tr>
<td>10.</td>
<td>Orange - Deskiñ (Lannion)</td>
<td>75.11</td>
</tr>
<tr>
<td>11.</td>
<td>RACAI (București)</td>
<td>74.67</td>
</tr>
<tr>
<td>12.</td>
<td>LyS-FASTPARSE (A Coruña)</td>
<td>74.42</td>
</tr>
</tbody>
</table>

Dozat and Manning
biLSTM + graph + tuning
(first-order features)

Model of Shi, Huang and Lee
biLSTM + transition + DP
(first-order features)
CoNLL 2017 Shared Task

Results: Unlabeled Attachment Score (UAS)

All treebanks

<table>
<thead>
<tr>
<th>Rank</th>
<th>Treebank</th>
<th>UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Stanford (Stanford)</td>
<td>81.30</td>
</tr>
<tr>
<td>2.</td>
<td>C2L2 (Ithaca)</td>
<td>80.35</td>
</tr>
<tr>
<td>3.</td>
<td>IMS (Stuttgart)</td>
<td>79.90</td>
</tr>
<tr>
<td>4.</td>
<td>HIT-SCIR (Harbin)</td>
<td>77.81</td>
</tr>
<tr>
<td>5.</td>
<td>LATTICE (Paris)</td>
<td>76.75</td>
</tr>
<tr>
<td>6.</td>
<td>NAIST SATO (Nara)</td>
<td>76.35</td>
</tr>
<tr>
<td>7.</td>
<td>UParse (Edinburgh)</td>
<td>75.49</td>
</tr>
<tr>
<td>8.</td>
<td>Koç University (İstanbul)</td>
<td>75.44</td>
</tr>
<tr>
<td>9.</td>
<td>ÚFAL - UDPipe 1.2 (Praha)</td>
<td>75.39</td>
</tr>
<tr>
<td>10.</td>
<td>TUL (Turku)</td>
<td>75.11</td>
</tr>
<tr>
<td>11.</td>
<td>École Polytechnique (Montreal)</td>
<td>74.67</td>
</tr>
<tr>
<td>12.</td>
<td>LyS-FASTPARSE (A Coruña)</td>
<td>74.42</td>
</tr>
</tbody>
</table>

- Dozat and Manning: biLSTM + graph + tuning (first-order features)
- Model of Shi, Huang and Lee: biLSTM + transition + DP (first-order features)

(both used also character-level LSTMs for words)
The best parsers in the world today are based on 1st-order decomposition over a BiLSTM.
The best parsers in the world today are based on 1st-order decomposition over a BiLSTM. I find this remarkable.
Take home questions

• Why does it work?
• What is encoded in these vectors?
• Where does it fail?
• How can we improve? (in an interesting way)?
  morphology? pre-training? multi-tasking? composition?
Take home questions

• Why does it work?
• What is encoded in these vectors?
• Where does it fail?
• How can we improve? (in an interesting way)?
  morphology? pre-training? multi-tasking? composition?

thanks for listening!