Natural Language Processing with Deep Learning

Tree Recursive Neural Networks, Constituency Parsing, and Sentiment
The spectrum of language in CS
Semantic interpretation of language – Not just word vectors

How can we work out the meaning of larger phrases?

- *The snowboarder is leaping over a mogul*
- *A person on a snowboard jumps into the air*

People interpret the meaning of larger text units – entities, descriptive terms, facts, arguments, stories – by **semantic composition** of smaller elements
Compositionality
Language understanding - & Artificial Intelligence - requires being able to understand bigger things from knowing about smaller parts
The Faculty of Language: What Is It, Who Has It, and How Did It Evolve?

Marc D. Hauser,1* Noam Chomsky,2 W. Tecumseh Fitch1

We argue that an understanding of the faculty of language requires substantial interdisciplinary cooperation. We suggest how current developments in linguistics can be profitably wedded to work in evolutionary biology, anthropology, psychology, and neuroscience. We submit that a distinction should be made between the faculty of language in the broad sense (FLB) and in the narrow sense (FLN). FLB includes a sensory-motor system, a conceptual-intentional system, and the computational mechanisms for recursion, providing the capacity to generate an infinite range of expressions from a finite set of elements. We hypothesize that FLN only includes recursion and is the only uniquely human component of the faculty of language. We further argue that FLN may have evolved for reasons other than language, hence comparative studies might look for evidence of such computations outside of the domain of communication (for example, number, navigation, and social relations).

If a martian graced our planet, it would be struck by one remarkable similarity among Earth’s living creatures and a key difference. Concerning similarity, it would note that all
Are languages recursive?

- Cognitively somewhat debatable (need to head to infinity)
- But: recursion is natural for describing language
  - [The person standing next to [the man from [the company that purchased [the firm that you used to work at]]]]
  - noun phrase containing a noun phrase containing a noun phrase
- It’s a very powerful prior for language structure
Penn Treebank tree

NP-SBJ
  | NNS  | VBD  | SBAR
  | Analysts, said -NONE-

NP-SBJ-1
  | NNP  | NNP  | VBZ
  | Mr. Stronach wants

VP
  | S
  | to

VP
  | TO
  | *-1

VP
  | VP
  | VBG
  | NP
  | PP-LOC
  | DT
  | a
  | RBR
  | JJ
  | role
  | IN
  | more
  | influential
  | -NONE-
  | VBG
  | NP
  | DT
  | running
  | NN
  | the
  | company
Building on Word Vector Space Models

How can we represent the meaning of longer phrases?

By mapping them into the same vector space!
How should we map phrases into a vector space?

Use principle of compositionality

The meaning (vector) of a sentence is determined by

1) the meanings of its words and
2) the rules that combine them.

Models in this section can jointly learn parse trees and compositional vector representations

Socher, Manning, and Ng. ICML, 2011
Constituency Sentence Parsing: What we want

The cat sat on the mat.
The cat sat on the mat.
Recursive vs. recurrent neural networks
Recursive vs. recurrent neural networks

- Recursive neural nets require a tree structure

- Recurrent neural nets cannot capture phrases without prefix context and often capture too much of last words in final vector
Recursive Neural Networks for Structure Prediction

Inputs: two candidate children’s representations
Outputs:
1. The semantic representation if the two nodes are merged.
2. Score of how plausible the new node would be.
Recursive Neural Network Definition

score = \begin{pmatrix} 1.3 \\ 8 \\ 5 \\ 3 \\ 3 \end{pmatrix} = \text{parent}

score = U^T p

p = \tanh(W \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + b),

Same \( W \) parameters at all nodes of the tree
The cat sat on the mat.
The cat sat on the mat.
The cat sat on the mat.
The cat sat on the mat.
Max-Margin Framework - Details

- The score of a tree is computed by the sum of the parsing decision scores at each node:

\[ s(x, y) = \sum_{n \in \text{nodes}(y)} s_n \]

- \( x \) is sentence; \( y \) is parse tree
Max-Margin Framework - Details

• Similar to max-margin parsing (Taskar et al. 2004), a supervised max-margin objective (maximize objective function)

\[
J = \sum_i s(x_i, y_i) - \max_{y \in A(x_i)} (s(x_i, y) + \Delta(y, y_i))
\]

\( y_i = \text{correct parse tree} \)

A: candidate parse tree

• The loss \( \Delta(y, y_i) \) penalizes all incorrect decisions

• Structure search for \( A(x) \) was greedy (join best nodes each time)
  • Instead: Beam search with chart
    *CKY parsing
Scene Parsing

Similar principle of compositionality.

- The meaning of a scene image is also a function of smaller regions,
- how they combine as parts to form larger objects,
- and how the objects interact.
Algorithm for Parsing Images

Same Recursive Neural Network as for natural language parsing! (Socher et al. ICML 2011)
## Multi-class segmentation

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel CRF (Gould et al., ICCV 2009)</td>
<td>74.3</td>
</tr>
<tr>
<td>Classifier on superpixel features</td>
<td>75.9</td>
</tr>
<tr>
<td>Region-based energy (Gould et al., ICCV 2009)</td>
<td>76.4</td>
</tr>
<tr>
<td>Local labelling (Tighe &amp; Lazebnik, ECCV 2010)</td>
<td>76.9</td>
</tr>
<tr>
<td>Superpixel MRF (Tighe &amp; Lazebnik, ECCV 2010)</td>
<td>77.5</td>
</tr>
<tr>
<td>Simultaneous MRF (Tighe &amp; Lazebnik, ECCV 2010)</td>
<td>77.5</td>
</tr>
<tr>
<td>Recursive Neural Network</td>
<td>78.1</td>
</tr>
</tbody>
</table>

Stanford Background Dataset (Gould et al. 2009)
Backpropagation Through Structure

Introduced by Goller & Küchler (1996)

Principally the same as general backpropagation

\[ \delta^{(l)} = \left( (W^{(l)})^T \delta^{(l+1)} \right) \circ f'(z^{(l)}) , \]
\[ \frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)} \]

Calculations resulting from the recursion and tree structure:

1. Sum derivatives of \( W \) from all nodes (like RNN)
2. Split derivatives at each node (for tree)
3. Add error messages from parent + node itself
BTS: Add error messages

- At each node:
  - What came up (fprop) must come down (bprop)
  - Total error messages = error messages from parent + error message from own score
BTS Python Code: forwardProp

```python
def forwardProp(self, node):
    # Recursion
    ...

    # This node's hidden activation
    node.h = np.dot(self.W, np.hstack([node.left.h, node.right.h])) + self.b
    # Relu
    node.h[node.h<0] = 0

    # Softmax
    node.probs = np.dot(self.Ws, node.h) + self.bs
    node.probs -= np.max(node.probs)
    node.probs = np.exp(node.probs)
    node.probs = node.probs / np.sum(node.probs)
```
BTS Python Code: backProp

def backProp(self, node, error=None):
    # Softmax grad
    deltas = node.probs
    deltas[node.label] -= 1.0
    self.dWs += np.outer(deltas, node.h)
    self.dbs += deltas
    deltas = np.dot(self.Ws.T, deltas)

    # Add deltas from above
    if error is not None:
        deltas += error

    # f'(z) now:
    deltas *= (node.h != 0)

    # Update word vectors if leaf node:
    if node.isLeaf:
        self.dL[node.word] += deltas
        return

    # Recursively backprop
    if not node.isLeaf:
        self.dW += np.outer(deltas, np.hstack([node.left.h, node.right.h]))
        self.db += deltas
        # Error signal to children
        deltas = np.dot(self.W.T, deltas)
        self.backProp(node.left, deltas[:self.hiddenDim])
        self.backProp(node.right, deltas[self.hiddenDim:])

\[
\delta^{(l)} = \left( (W^{(l)})^T \delta^{(l+1)} \right) \circ f'(z^{(l)}),
\]

\[
\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}
\]
Discussion: Simple TreeRNN

• Decent results with single matrix TreeRNN

• Single weight matrix TreeRNN could capture some phenomena but not adequate for more complex, higher order composition and parsing long sentences

• There is no real interaction between the input words

• The composition function is the same for all syntactic categories, punctuation, etc.
Syntactically-Untied RNN

- A symbolic Context-Free Grammar (CFG) backbone is adequate for basic syntactic structure
- We use the discrete syntactic categories of the children to choose the composition matrix
- A TreeRNN can do better with different composition matrix for different syntactic environments
- The result gives us a better semantics

[Socher, Bauer, Manning, Ng 2013]
Compositional Vector Grammars

• Problem: Speed. Every candidate score in beam search needs a matrix-vector product.

• Solution: Compute score only for a subset of trees coming from a simpler, faster model (PCFG)
  • Prunes very unlikely candidates for speed
  • Provides coarse syntactic categories of the children for each beam candidate

• Compositional Vector Grammar = PCFG + TreeRNN
Related Work for parsing

- Resulting CVG Parser is related to previous work that extends PCFG parsers
- Klein and Manning (2003a) : manual feature engineering
- Petrov et al. (2006) : learning algorithm that splits and merges syntactic categories
- Lexicalized parsers (Collins, 2003; Charniak, 2000): describe each category with a lexical item
- Hall and Klein (2012) combine several such annotation schemes in a factored parser.
- CVGs extend these ideas from discrete representations to richer continuous ones
Experiments

- Standard WSJ split, labeled F1
- Based on simple PCFG with fewer states
- Fast pruning of search space, few matrix-vector products
- 3.8% higher F1

<table>
<thead>
<tr>
<th>Parser</th>
<th>Test, All Sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stanford PCFG, (Klein and Manning, 2003a)</td>
<td>85.5</td>
</tr>
<tr>
<td>Stanford Factored (Klein and Manning, 2003b)</td>
<td>86.6</td>
</tr>
<tr>
<td>Factored PCFGs (Hall and Klein, 2012)</td>
<td>89.4</td>
</tr>
<tr>
<td>Collins (Collins, 1997)</td>
<td>87.7</td>
</tr>
<tr>
<td>SSN (Henderson, 2004)</td>
<td>89.4</td>
</tr>
<tr>
<td>Berkeley Parser (Petrov and Klein, 2007)</td>
<td>90.1</td>
</tr>
<tr>
<td>CVG (RNN) (Socher et al., ACL 2013)</td>
<td>85.0</td>
</tr>
<tr>
<td>CVG (SU-RNN) (Socher et al., ACL 2013)</td>
<td>90.4</td>
</tr>
<tr>
<td>Charniak - Self Trained (McClosky et al. 2006)</td>
<td>91.0</td>
</tr>
<tr>
<td>Charniak - Self Trained-ReRanked (McClosky et al. 2006)</td>
<td>92.1</td>
</tr>
</tbody>
</table>
Analysis of resulting vector representations

All the figures are adjusted for seasonal variations
1. All the numbers are adjusted for seasonal fluctuations
2. All the figures are adjusted to remove usual seasonal patterns

Knight-Ridder wouldn’t comment on the offer
1. Harsco declined to say what country placed the order
2. Coastal wouldn’t disclose the terms

Sales grew almost 7% to $UNK m. from $UNK m.
1. Sales rose more than 7% to $94.9 m. from $88.3 m.
2. Sales surged 40% to UNK b. yen from UNK b.
Improving Deep Learning Semantic Representations using a TreeLSTM
[Tai et al., ACL 2015; also Zhu et al. ICML 2015]

Goals:

• Still trying to represent the meaning of a sentence as a location in a (high-dimensional, continuous) vector space
• In a way that accurately handles semantic composition and sentence meaning
• Generalizing the widely used chain-structured LSTM to trees
Long Short-Term Memory (LSTM) Units for Sequential Composition

Gates are vectors in $[0,1]^d$ multiplied element-wise for soft masking.
Tree-Structured Long Short-Term Memory Networks

[Tai et al., ACL 2015]
Tree-structured LSTM

Generalizes sequential LSTM to trees with any branching factor
## Results: Sentiment Analysis: Stanford Sentiment Treebank

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy % (Fine-grain, 5 classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNTN (Socher et al. 2013)</td>
<td>45.7</td>
</tr>
<tr>
<td>Paragraph-Vec (Le &amp; Mikolov 2014)</td>
<td>48.7</td>
</tr>
<tr>
<td>DRNN (Irsoy &amp; Cardie 2014)</td>
<td>49.8</td>
</tr>
<tr>
<td>LSTM</td>
<td>46.4</td>
</tr>
<tr>
<td>Tree LSTM (this work)</td>
<td>50.9</td>
</tr>
</tbody>
</table>
## Results: Sentiment Analysis: Stanford Sentiment Treebank

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy % (Pos/Neg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNTN (Socher et al. 2013)</td>
<td>85.4</td>
</tr>
<tr>
<td>Paragraph-Vec (Le &amp; Mikolov 2014)</td>
<td>87.8</td>
</tr>
<tr>
<td>DRNN (Irsoy &amp; Cardie 2014)</td>
<td>86.6</td>
</tr>
<tr>
<td>LSTM</td>
<td>84.9</td>
</tr>
<tr>
<td>Tree LSTM (this work)</td>
<td>88.0</td>
</tr>
</tbody>
</table>
Forget Gates: Selective State Preservation

- Stripes = forget gate activations; more white ⇒ more preserved

```
a  waste

of

good  performances
```
Tree-to-tree Neural Networks for Program Translation
[Chen, Liu, and Song NeurIPS 2018]

- Explores using tree-structured encoding and generation for translation between programming languages.
- In generation, you use attention over the source tree.

CoffeeScript Program: x=1 if y==0

```plaintext
Parse Tree

Block
  If
    Op==
      Value
        Identifier Literal
          y
      Value
        Number Literal
          0
    Block
      Assign
        Value
          Identifier Literal
            x
        Value
          Number Literal
            1
```

JavaScript Program: if (y === 0) { x = 1; }

```plaintext
Parse Tree

Program
  IfStatement
    BinaryExpression
      Identifier
        y
      ===
      Literal
        0
      AssignExpression
        Identifier
          x
        =
        Literal
          1
```

*tree LSTM encoder decoder*
Tree-to-tree Neural Networks for Program Translation
[Chen, Liu, and Song NeurIPS 2018]

*T: parse tree token sequence
P: raw program token sequence

<table>
<thead>
<tr>
<th></th>
<th>Tree2tree</th>
<th>Seq2seq</th>
<th>Seq2tree</th>
<th>Tree2seq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T→T</td>
<td>T→T</td>
<td>T→T</td>
<td>T→T</td>
</tr>
<tr>
<td></td>
<td>(-PF)</td>
<td>(-Attn)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P→P</td>
<td>P→T</td>
<td>T→P</td>
<td>T→T</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CoffeeScript to JavaScript translation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CJ-AS</td>
<td>99.57%</td>
<td>98.80%</td>
<td>0.09%</td>
<td>90.51%</td>
</tr>
<tr>
<td>CJ-BS</td>
<td>99.75%</td>
<td>99.67%</td>
<td>0%</td>
<td>97.44%</td>
</tr>
<tr>
<td>CJ-AL</td>
<td>97.15%</td>
<td>71.52%</td>
<td>0%</td>
<td>21.04%</td>
</tr>
<tr>
<td>CJ-BL</td>
<td>95.60%</td>
<td>78.61%</td>
<td>0%</td>
<td>19.26%</td>
</tr>
</tbody>
</table>

JavaScript to CoffeeScript translation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>JC-AS</td>
<td>87.75%</td>
<td>85.11%</td>
<td>0.09%</td>
<td>83.07%</td>
</tr>
<tr>
<td>JC-BS</td>
<td>86.37%</td>
<td>80.35%</td>
<td>0%</td>
<td>80.49%</td>
</tr>
<tr>
<td>JC-AL</td>
<td>78.59%</td>
<td>54.93%</td>
<td>0%</td>
<td>77.10%</td>
</tr>
<tr>
<td>JC-BL</td>
<td>75.62%</td>
<td>44.40%</td>
<td>0%</td>
<td>73.14%</td>
</tr>
</tbody>
</table>

*A, B: program segments; S, L: length