Natural Language Processing with Deep Learning
CS224N/Ling284

Christopher Manning
Lecture 18: Tree Recursive Neural Networks, Constituency Parsing, and Sentiment
1. 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, Noam Chomsky, W. Tecumseh Fitch

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
Analysts said Mr. Stronach wants to resume a more influential role in the company.
2. 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.

Socher, Manning, and Ng. ICML, 2011

Models in this section can jointly learn parse trees and compositional vector representations
Constituency Sentence Parsing: What we want

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

```
the   country   of   my   birth
0.4   0.3   2.1   3.3   7   7   4   4.5   2.3   3.6
1   3.5
1   5
0.4   2.1   3.3   7   7   4   4.5   2.3   3.6
1   3.5
1   5
4   4.5
2.5   3.8
```

17
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

\[ score = 1.3 \]

= parent

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

Same \( W \) parameters at all nodes of the tree
Parsing a sentence with an RNN (greedily)

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

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

• 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
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)
3. 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: 1) Sum derivatives of all nodes

You can actually assume it’s a different \( W \) at each node

Intuition via example:

\[
\frac{\partial}{\partial W} f(W(f(Wx)))
\]

\[
= f'(W(f(Wx))) \left( \left( \frac{\partial}{\partial W} W \right) f(Wx) + W \frac{\partial}{\partial W} f(Wx) \right)
\]

\[
= f'(W(f(Wx))) (f(Wx) + W f'(Wx)x)
\]

If we take separate derivatives of each occurrence, we get same:

\[
\frac{\partial}{\partial W_2} f(W_2(f(W_1x))) + \frac{\partial}{\partial W_1} f(W_2(f(W_1x)))
\]

\[
= f'(W_2(f(W_1x))) (f(W_1x)) + f'(W_2(f(W_1x))) (W_2 f'(W_1x)x)
\]

\[
= f'(W_2(f(W_1x))) (f(W_1x) + W_2 f'(W_1x)x)
\]

\[
= f'(W(f(Wx))) (f(Wx) + W f'(Wx)x)
\]
BTS: 2) Split derivatives at each node

During forward prop, the parent is computed using 2 children

\[
p = \tanh(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b)
\]

Hence, the errors need to be computed wrt each of them:

\[
\delta_{p \rightarrow c_1 c_2} = [\delta_{p \rightarrow c_1} \ \delta_{p \rightarrow c_2}]
\]

where each child’s error is \( n \)-dimensional
BTS: 3) 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
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)
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.

[Sohcer, Bauer, Manning, Ng 2013]

- 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
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>
SU-RNN / CVG [Socher, Bauer, Manning, Ng 2013]

Learns soft notion of head words

Initialization: \[ W(\cdot) = 0.5[I_{n \times n} I_{n \times n} 0_{n \times 1}] + \epsilon \]
SU-RNN / CVG [Socher, Bauer, Manning, Ng 2013]
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.
One way to make the composition function more powerful was by untying the weights $W$.

But what if words act mostly as an operator, e.g. “very” in “very good”?

Proposal: A new composition function
Compositionality Through Recursive Matrix-Vector Recursive Neural Networks

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

Recursive Matrix-Vector Model

\[
f(Ba, Ab) = \tanh(W \begin{pmatrix} C_2 & C_1 \\ C_1 & C_2 \end{pmatrix} + b)
\]

- vector
- matrix

\[
Ba = \begin{pmatrix} a, A \end{pmatrix}
\]

\[
Ab = \begin{pmatrix} b, B \end{pmatrix}
\]

\[
movie = \begin{pmatrix} c, C \end{pmatrix}
\]
Matrix-vector RNNs
[Socher, Huval, Bhat, Manning, & Ng, 2012]

\[
p = f \left( W \begin{bmatrix} B_a \\ A_b \end{bmatrix} \right)
\]

\[
P = g(A, B) = W_M \begin{bmatrix} A \\ B \end{bmatrix}
\]

\[W_M \in \mathbb{R}^{n \times 2n}\]
Predicting Sentiment Distributions

Good example for non-linearity in language
Classification of Semantic Relationships

• Can an MV-RNN learn how a large syntactic context conveys a semantic relationship?

• My \([\text{apartment}]_{e1}\) has a pretty large \([\text{kitchen}]_{e2}\) \(\rightarrow\) component-whole relationship \((e2,e1)\)

• Build a single compositional semantics for the minimal constituent including both terms
# Classification of Semantic Relationships

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Features</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>POS, stemming, syntactic patterns</td>
<td>60.1</td>
</tr>
<tr>
<td>MaxEnt</td>
<td>POS, WordNet, morphological features, noun compound system, thesauri, Google n-grams</td>
<td>77.6</td>
</tr>
<tr>
<td>SVM</td>
<td>POS, WordNet, prefixes, morphological features, dependency parse features, Levin classes, PropBank, FrameNet, NomLex-Plus, Google n-grams, paraphrases, TextRunner</td>
<td>82.2</td>
</tr>
<tr>
<td>RNN</td>
<td>–</td>
<td>74.8</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>–</td>
<td>79.1</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>POS, WordNet, NER</td>
<td>82.4</td>
</tr>
</tbody>
</table>
Version 4: Recursive Neural Tensor Network
Socher, Perelygin, Wu, Chuang, Manning, Ng, and Potts 2013

• Less parameters than MV-RNN
• Allows the two word or phrase vectors to interact multiplicatively

\[ p_2 = g(a, p_1) \]
\[ p_1 = g(b, c) \]

Neural Tensor Layer

\[ p = f \]
\[ p = f \begin{pmatrix} b & c \end{pmatrix}^{T} + b + c + w \begin{pmatrix} b \\ c \end{pmatrix} \]
Beyond the bag of words: Sentiment detection

Is the tone of a piece of text positive, negative, or neutral?

- Sentiment is that sentiment is “easy”
- Detection accuracy for longer documents ~90%, BUT

... ... loved ... ... ... ... great ... ... ... ... ... ... impressed
... ... ... ... ... ... marvelous ... ... ... ... ... ... 

With this cast, and this subject matter, the movie should have been funnier and more entertaining.
Stanford Sentiment Treebank

- 215,154 phrases labeled in 11,855 sentences
- Can actually train and test compositions

http://nlp.stanford.edu:8080/sentiment/
Better Dataset Helped All Models

- Hard negation cases are still mostly incorrect
- We also need a more powerful model!
Version 4: Recursive Neural Tensor Network

Idea: Allow both additive and mediated multiplicative interactions of vectors
Recursive Neural Tensor Network
Recursive Neural Tensor Network

\[ p_2 = g(a, p_1) \]

\[ p_1 = g(b, c) \]

\[ \begin{pmatrix} b^T \end{pmatrix}_{[1:2]} \begin{pmatrix} b \\ c \end{pmatrix} + W \begin{pmatrix} b \\ c \end{pmatrix} \]
Recursive Neural Tensor Network

- Use resulting vectors in tree as input to a classifier like logistic regression
- Train all weights jointly with gradient descent
Positive/Negative Results on Treebank

Classifying Sentences: Accuracy improves to 85.4
Experimental Results on Treebank

- RNTN can capture constructions like $X$ but $Y$
- RNTN accuracy of 72%, compared to MV-RNN (65%), biword NB (58%) and RNN (54%)
Negation Results

When negating negatives, positive activation should increase!

Demo: http://nlp.stanford.edu:8080/sentiment/
Version 5:
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
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>
<tr>
<td>Method</td>
<td>Accuracy % (Pos/Neg)</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----------------------</td>
</tr>
<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>
# Results: Semantic Relatedness

**SICK 2014** (Sentences Involving Compositional Knowledge)

<table>
<thead>
<tr>
<th>Method</th>
<th>Pearson correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word vector average</td>
<td>0.758</td>
</tr>
<tr>
<td>Meaning Factory (Bjerva et al. 2014)</td>
<td>0.827</td>
</tr>
<tr>
<td>ECNU (Zhao et al. 2014)</td>
<td>0.841</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.853</td>
</tr>
<tr>
<td>Tree LSTM</td>
<td><strong>0.868</strong></td>
</tr>
</tbody>
</table>
Forget Gates: Selective State Preservation

- Stripes = forget gate activations; more white $\Rightarrow$ more preserved
5. QCD-Aware Recursive Neural Networks for Jet Physics
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

```
Parse Tree
```

```
<table>
<thead>
<tr>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>If</td>
</tr>
<tr>
<td>Op==</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier Literal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assign</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>Identifier Literal</td>
</tr>
<tr>
<td>Number Literal</td>
</tr>
</tbody>
</table>

| x |

| 1 |

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

```
Parse Tree
```

```
<table>
<thead>
<tr>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>IfStatement</td>
</tr>
<tr>
<td>BinaryExpression</td>
</tr>
<tr>
<td>Identifier</td>
</tr>
<tr>
<td>===</td>
</tr>
<tr>
<td>Literal</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ExpressionStatement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlockStatement</td>
</tr>
<tr>
<td>AssignExpression</td>
</tr>
</tbody>
</table>

| x |
| 1 |
# Tree-to-tree Neural Networks for Program Translation

[Chen, Liu, and Song NeurIPS 2018]

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

**CoffeeScript to JavaScript translation**

<table>
<thead>
<tr>
<th></th>
<th>Tree2tree</th>
<th>Seq2seq</th>
<th>Seq2tree</th>
<th>Tree2seq</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></td>
<td>99.75%</td>
<td>99.67%</td>
<td>0%</td>
<td>97.44%</td>
</tr>
<tr>
<td>CJ-BL</td>
<td>97.15%</td>
<td>71.52%</td>
<td>0%</td>
<td>21.04%</td>
</tr>
<tr>
<td></td>
<td>95.60%</td>
<td>78.61%</td>
<td>0%</td>
<td>19.26%</td>
</tr>
</tbody>
</table>

**JavaScript to CoffeeScript translation**

<table>
<thead>
<tr>
<th></th>
<th>Tree2tree</th>
<th>Seq2seq</th>
<th>Seq2tree</th>
<th>Tree2seq</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></td>
<td>86.37%</td>
<td>80.35%</td>
<td>0%</td>
<td>80.49%</td>
</tr>
<tr>
<td>JC-BL</td>
<td>78.59%</td>
<td>54.93%</td>
<td>0%</td>
<td>77.10%</td>
</tr>
<tr>
<td></td>
<td>75.62%</td>
<td>44.40%</td>
<td>0%</td>
<td>73.14%</td>
</tr>
</tbody>
</table>
### Tree-to-tree Neural Networks for Program Translation

[Chen, Liu, and Song NeurIPS 2018]

<table>
<thead>
<tr>
<th></th>
<th>Tree2tree</th>
<th>J2C#</th>
<th>1pSMT</th>
<th>mppSMT Reported in [22]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucene</td>
<td>72.8%</td>
<td>21.5%</td>
<td>21.6%</td>
<td>40.0%</td>
</tr>
<tr>
<td>POI</td>
<td>72.2%</td>
<td>18.9%</td>
<td>34.6%</td>
<td>48.2%</td>
</tr>
<tr>
<td>Itext</td>
<td>67.5%</td>
<td>25.1%</td>
<td>24.4%</td>
<td>40.6%</td>
</tr>
<tr>
<td>JGit</td>
<td>68.7%</td>
<td>10.7%</td>
<td>23.0%</td>
<td>48.5%</td>
</tr>
<tr>
<td>JTS</td>
<td>68.2%</td>
<td>11.7%</td>
<td>18.5%</td>
<td>26.3%</td>
</tr>
<tr>
<td>Antlr</td>
<td>31.9% (58.3%)</td>
<td>10.0%</td>
<td>11.5%</td>
<td>49.1%</td>
</tr>
</tbody>
</table>