**Summary**

- **RNTN** has been previously successfully applied to sentiment analysis. However, its training process is time-consuming.
- Our objective is to speed up the training of RNTNs with mini-batch gradient descent, which involves many matrix-vector multiplications.
- The existing code performs forward and backward propagation one example at a time, and the matrix-vector multiplication is executed one node at a time.
- We observe that the weight matrix is shared across training examples and across nodes, making it possible to combine many matrix-vector multiplications into one large matrix-matrix multiplication.
- We have implemented our idea. As a result, the training time has been significantly reduced.

**Recursive Neural Tensor Network (RNTN)**

- Each training example has its own parse-tree structure.

![Diagram of a tree structured RNTN](image)

- Parameters \( \theta = (L, W, W_v, V) \):
  - \( L \): word embedding matrix.
  - \( W \): weight matrix.
  - \( W_v \): classification matrix.
  - \( V \): weight tensor.
- Node vectors are computed starting from the leaf nodes.
  - Representation at node \( i \):
    \[ x_i' = \tanh \left( \frac{x_i}{x_R} \right)^T V \left( \frac{x_i}{x_R} \right) + W \left( \frac{x_i}{x_R} \right) \in \mathbb{R}^d, \]
  - Soft classification at node \( x_i' \):
    \[ y_i' = \text{softmax}(W_i x_i). \]
  - Each node \( x_i' \) has a target vector \( t_i' \).
- Error function \( E(\theta) = -\sum_i \log y_i' + \lambda ||\theta||^2 \).
- **Training:**
  - Data set: Stanford Sentiment Treebank
  - Batched Adaptive Gradient Descent (AdaGrad)
  - Existing code: CoreNLP

**Forward Propagation**

- **Grouping Nodes Across Trees (assume batch size = 3)**
- **Step 1:**
  - \( S' = \sum_{i=1}^{d} \delta_{s_i} \text{com} \cdot S' + S' \cdot f'(x) \)
  - where \( \delta_{s_i} \text{com} : \) Hadamard product, \( f'(x) = 1 - x^2 \).
- **Step 2:**
  - \( x_i' \)
  - **Step 3:** Traverse up until all nodes are visited.

**Grouping \( k \) pairs of nodes**

- Concatenation vectors: let \( u_i = [x_i, x_i'] \in \mathbb{R}^{2d}, i = 1, \ldots, k \)
- Grouping matrix-vector multiplication:
  \[ W_{u_1} W_{u_2} \cdots W_{u_k} = W [u_1 u_2 \cdots u_k]. \]
- Grouping tensor-vector operation:
  \[ V \in \mathbb{R}^{2d \times d \times d} \]
  \[ [u_1 V u_2 V u_3 \cdots u_k V] \in \mathbb{R}^{d \times k} \]

**Backward Propagation**

- **Formula for errors \( \delta_i \) at node \( i \).**
  - \( \delta_i \text{com} = \delta_i \text{com} \cdot \text{down} \) if \( x_i' \) is the root
  - \( \delta_i \text{com} = \delta_i \text{com} \cdot \text{down} \) if \( x_i' \) is the left child of \( x_i \)
  - \( \delta_i \text{com} = \delta_i \text{com} \cdot \text{down} \) if \( x_i' \) is the right child of \( x_i \)
  - \( \delta_i \text{com} = \delta_i \text{com} + S' \cdot f'(x) \)

**Results**

- **Word Vector size = 50.**

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Training Time Per Epoch (sec)</th>
<th>Before</th>
<th>After</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1529</td>
<td>1012</td>
<td></td>
<td>33.8%</td>
</tr>
<tr>
<td>50</td>
<td>1532</td>
<td>975</td>
<td></td>
<td>36.4%</td>
</tr>
<tr>
<td>100</td>
<td>1567</td>
<td>9650</td>
<td></td>
<td>38.4%</td>
</tr>
</tbody>
</table>

- **Batch size = 50.**

<table>
<thead>
<tr>
<th>Word Vector Size</th>
<th>Training Time Per Epoch (sec)</th>
<th>Before</th>
<th>After</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>191</td>
<td>136</td>
<td></td>
<td>28.8%</td>
</tr>
<tr>
<td>50</td>
<td>1533</td>
<td>975</td>
<td></td>
<td>36.4%</td>
</tr>
<tr>
<td>100</td>
<td>12711</td>
<td>7828</td>
<td></td>
<td>38.4%</td>
</tr>
</tbody>
</table>

- Our code is at [https://github.com/avati/CoreNLP/tree/tnnt](https://github.com/avati/CoreNLP/tree/tnnt)