Distributed Representations of Words and Phrases and their Compositionality
Agenda

- Skip gram model
- Hierarchical Softmax
- Negative Sampling
- Subsampling of Frequent Words
- Empirical Results
- Learning Phrases
- Conclusion
Skip Gram Model

- Objective:
  
  “find word representations that are useful for predicting the surrounding words in a sentence”

- C: size of the training context (which can be a function of the center word \(w_t\)). Larger the \(c\) more accurate is the output

\[
\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)
\]
Hierarchical Softmax

- Softmax function:

\[
p(w_o|w_i) = \frac{\exp\left( v'_w o \top v_{w_i} \right)}{\sum_{w=1}^{W} \exp\left( v'_w \top v_{w_i} \right)}
\]

- Hierarchical Softmax function:

log2 (W) nodes
Negative Sampling

- Objective: Each training sample only modify a small percentage of the weights and not all of them.
- "negative" word is one for which we want the network to output a 0.
- Still update the weights for our "positive" word.
- Selecting 5-20 words - smaller dataset.
- 2-5 words - larger dataset.
Subsampling Of Frequent Words

- The vector representations of frequent words do not change significantly after training on several million examples.
- Each word $w_i$ in the training set is discarded with probability computed by the formula:

\[
P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}
\]

- where $f(w_i)$ is the frequency of word $w_i$ and $t$ is a chosen threshold, typically around $10^{-5}$.
Empirical Results

- Finding analogies such as “Germany” : “Berlin” :: “France” : ?, which are solved by finding a vector $x$ such that $\text{vec}(x)$ closest to

$$\text{vec(“Berlin”)} - \text{vec(“Germany”)} + \text{vec(“France”)}$$

- The task has two categories:
  - syntactic analogies (such as “quick” : “quickly” :: “slow” : “slowly”)
  - semantic analogies, such as the country to capital city relationship.
## Empirical Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Time [min]</th>
<th>Syntactic [%]</th>
<th>Semantic [%]</th>
<th>Total accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG-5</td>
<td>38</td>
<td>63</td>
<td>54</td>
<td>59</td>
</tr>
<tr>
<td>NEG-15</td>
<td>97</td>
<td>63</td>
<td>58</td>
<td>61</td>
</tr>
<tr>
<td>HS-Huffman</td>
<td>41</td>
<td>53</td>
<td>40</td>
<td>47</td>
</tr>
<tr>
<td>NCE-5</td>
<td>38</td>
<td>60</td>
<td>45</td>
<td>53</td>
</tr>
</tbody>
</table>

The following results use $10^{-6}$ subsampling:

<table>
<thead>
<tr>
<th>Method</th>
<th>Time [min]</th>
<th>Syntactic [%]</th>
<th>Semantic [%]</th>
<th>Total accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG-5</td>
<td>14</td>
<td>61</td>
<td>58</td>
<td>60</td>
</tr>
<tr>
<td>NEG-15</td>
<td>36</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>HS-Huffman</td>
<td>21</td>
<td>52</td>
<td>59</td>
<td>55</td>
</tr>
</tbody>
</table>
Learning Phrases

- To learn vector representation for phrases, find words that appear frequently together, and infrequently in other contexts.
- For example, “New York Times” and “Toronto Maple Leafs” are replaced by unique tokens in the training data, while a bigram “this is” will remain unchanged.

\[
\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}.
\]

- \(\delta\) is used as a discounting coefficient and prevents too many phrases consisting of very infrequent words to be formed.
- The bigrams with score above the chosen threshold are selected as phrases.
Conclusion

- The subsampling method enhances performance by 2X-10X factor.
- Negative sampling algorithm, which is an extremely simple training method that learns accurate representations specially for frequent words.
- Subsampling of the frequent words results in both faster training and significantly better representations of uncommon words.
- Learning phrases with skip-gram model is being used in many applications and show very positive results.