Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean

Presented by: Jamsrandorj Dagvadorj
Outline

- Introduction
- The Skip-gram model
- Contribution
- Conclusion
Introduction

- Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words
  - Simplicity
  - Robustness
  - Observation
- With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models
Introduction

- Distributed representations of words in a vector space
- Many of linguistic regularities and patterns can be represented as linear translations

Example:
vec(“Madrid”) - vec(“Spain”) + vec(“France”) is closer to vec(“Paris”) than any other word vectors
The Skip-gram model

- An efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships
- Training of the Skip-gram model does not involve dense matrix multiplications
- An optimized single-machine implementation can train on more than 100 billion words in one day
The Skip-gram model

To maximize the average log probability

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

\( T \): the number of words  
\( w_1, w_2, w_3, ..., w_T \): a sequence of words  
\( c \): the size of the training context

The Skip-gram model architecture
Contribution

- Negative Sampling vs Hierarchical Softmax
- Subsampling of Frequent Words
- Learning Phrase
- Additive Compositionality
Negative Sampling vs **Hierarchical Softmax**

- Hierarchical Softmax: A computationally efficient approximation of the full softmax
- It uses a binary tree representation of the output layer with the $W$ words as its leaves and, for each node, explicitly represent the relative probabilities of its child nodes
Negative Sampling vs Hierarchical Softmax

\[ \log \sigma(v'_w_0 \top v_{w_l}) + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[ \log \sigma(-v'_w_i \top v_{w_l}) \right] \]

- \( k \): negative samples
- \( P_n(w) \): the noise distribution

replace every \( P(w_0 | w_l) \) term in the Skip-gram objective.
Subsampling of Frequent Words

- The most frequent words can easily occur hundreds of millions times (e.g., “in”, “the”, and “a”)
- Usually provide less information value than the rare words
- The **imbalance** between the rare and frequent words

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

\[
f(w_i) : \text{the frequency of word } w_i
\]

\[
t : \text{a threshold } (~ 10^5)
\]
Empirical Result

- The analogical reasoning task [1]
  - The closest word vector to \( \text{vec(“Berlin”) - vec(“Germany”) + vec(“France”) = ?} \)
  - Correct if \( \text{vec(“Paris”) is found} \)
- Syntactic
  - \( \text{vec(“quickly”) - vec(“quick”) + vec(“slow”) \ ? vec(“slowly”) } \)
- Semantic
  - Example: country to capital city relationship

## Empirical Result

<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 $10^{-5}$ 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 Phrase

- Many phrases have a meaning that is not a simple composition of the meanings of its individual words.
  - “New York Times” is not “New” + “York” + “Times”
- Find words that appear frequently together, and infrequently in other contexts
  - “New York Times” -> 1 token
  - “This is” -> remain unchanged
- A simple data-driven approach:

\[
score(w_i, w_j) = \frac{score(w_i,w_j)-\delta}{score(w_i) * score(w_j)}
\]

\(\delta : a\ discounting\ coefficient\)
### Newspaper:

|----------|-------------------------|-------------------------------|-----------|--------------------------|---------------------|

### NHL Teams:

<table>
<thead>
<tr>
<th>Boston</th>
<th>Boston Bruins Phoenix</th>
<th>Phoenix Coyotes</th>
<th>Montreal</th>
<th>Montreal Canadiens Nashville</th>
<th>Nashville Predators</th>
</tr>
</thead>
</table>

### NBA Teams:

<table>
<thead>
<tr>
<th>Detroit</th>
<th>Detroit Pistons Oakland</th>
<th>Golden State Warriors</th>
<th>Toronto</th>
<th>Toronto Raptors Memphis</th>
<th>Memphis Grizzlies</th>
</tr>
</thead>
</table>

### Airlines:

<table>
<thead>
<tr>
<th>Austria</th>
<th>Austrian Airlines Belgium</th>
<th>Brussels Airlines</th>
<th>Spain</th>
<th>Spainair Greece</th>
<th>Aegean Airlines</th>
</tr>
</thead>
</table>

### Company executives:

<table>
<thead>
<tr>
<th>Steve Ballmer</th>
<th>Microsoft Steve Ballmer</th>
<th>Larry Page</th>
<th>Google</th>
<th>Larry Page</th>
<th>Google</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samuel J. Palmisano</td>
<td>IBM Samuel J. Palmisano</td>
<td>Werner Vogels</td>
<td>Amazon</td>
<td>Werner Vogels</td>
<td>Amazon</td>
</tr>
</tbody>
</table>
## Phrase Skip-Gram Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Dimensionality</th>
<th>No subsampling [%]</th>
<th>$10^{-5}$ subsampling [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG-5</td>
<td>300</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>NEG-15</td>
<td>300</td>
<td>27</td>
<td>42</td>
</tr>
<tr>
<td>HS-Huffman</td>
<td>300</td>
<td>19</td>
<td>47</td>
</tr>
</tbody>
</table>

High accuracy: 72%
Dimensionality: 1000
Training words: 33 billion
Additive Compositionality

- The word and phrase representations exhibit a linear structure
- Possible to meaningfully combine words by an element-wise addition of their vector representations

<table>
<thead>
<tr>
<th>Czech + currency</th>
<th>Vietnam + capital</th>
<th>German + airlines</th>
<th>Russian + river</th>
<th>French + actress</th>
</tr>
</thead>
<tbody>
<tr>
<td>koruna</td>
<td>Hanoi</td>
<td>airline Lufthansa</td>
<td>Moscow</td>
<td>Juliette Binoche</td>
</tr>
<tr>
<td>Check crown</td>
<td>Ho Chi Minh City</td>
<td>carrier Lufthansa</td>
<td>Volga River</td>
<td>Vanessa Paradis</td>
</tr>
<tr>
<td>Polish zolty</td>
<td>Viet Nam</td>
<td>flag carrier Lufthansa</td>
<td>upriver</td>
<td>Charlotte Gainsbourg</td>
</tr>
<tr>
<td>CTK</td>
<td>Vietnamese</td>
<td>Lufthansa</td>
<td>Russia</td>
<td>Cecile De</td>
</tr>
</tbody>
</table>
Comparison to Published Word Representations

<table>
<thead>
<tr>
<th>Model (training time)</th>
<th>Redmond</th>
<th>Havel</th>
<th>ninjutsu</th>
<th>graffiti</th>
<th>capitulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collobert (50d) (2 months)</td>
<td>conyers lubock keene</td>
<td>plauen dzerzhinsky oesterreich</td>
<td>reiki kohona karate</td>
<td>cheesecake gossip dioramas</td>
<td>abdicate accede rearm</td>
</tr>
<tr>
<td>Turian (200d) (few weeks)</td>
<td>McCarthy Alston Cousins</td>
<td>Jewell Arzu Ovitz</td>
<td>-</td>
<td>gunfire emotion impunity</td>
<td>-</td>
</tr>
<tr>
<td>Mnih (100d) (7 days)</td>
<td>Podhurst Harlang Agarwal</td>
<td>Pontiff Pinochet Rodionov</td>
<td>-</td>
<td>anaesthetics monkeys Jews</td>
<td>Mavericks planning hesitated</td>
</tr>
<tr>
<td>Skip-Phrase (1000d, 1 day)</td>
<td>Redmond Wash. Redmond Washington Microsoft</td>
<td>Vaclav Havel president Vaclav Havel Velvet Revolution</td>
<td>ninja martial arts swordsmanship</td>
<td>spray paint graffitti taggers</td>
<td>capitulation capitulated capitulating</td>
</tr>
</tbody>
</table>

Training words: 30 billion words (~2 to 3 orders of magnitude more data)
Training time: 1 day
Conclusion

- Computationally efficient model architecture
  - Negative sampling algorithm
  - Subsampling of frequent words
- Linear Structure
  - Vector Addition
  - Element-wise addition
- Learning Phrases

Source code: https://code.google.com/archive/p/word2vec/
Thank you