word2vec

Anđelka Zečević
andjelkaz@matf.bg.ac.rs
word2vec

Tool for embedded word representation generation according to the results of:


https://code.google.com/archive/p/word2vec/
overview

● Motivation:
  ○ How do we handle semantics?
  ○ How can we represent words in order to keep the meaning?

● Neural Network Language Model
  ○ Continuous Bag of Words
  ○ Skip-gram
A bottle of tesgüino is on the table.
Everybody likes tesgüino.
Tesgüino makes you drunk.
We make tesgüino out of corn.

The meaning of a word is related to the distribution of the words around it.

from *Speech and Language Processing* by Dan Jurafsky and James H. Martin.
*Tesgüino* is a corn beer made by the Tarahumara Indians of Sierra Madre in Mexico.
distributed semantics

The hypothesis of linguistics by Firth (1957):

“We shall know the word by the company it keeps.”

There can be many types of relatedness:

- synonyms: big and large
- concept categories: dog, cat → animals
- associations: bee & honey
- analogies: big and bigger as small and smaller
- ...
distributed semantics = vector semantics

Words as one-hot vectors:

<table>
<thead>
<tr>
<th></th>
<th>“a”</th>
<th>“abbreviations”</th>
<th>“zoology”</th>
<th>“zoom”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

No semantics!

The size of one-hot vector is equal to the vocabulary size.
distributed semantics = vector semantics

Words as rows of term-document matrix:

\[
\begin{array}{ccccc}
& D1 & D2 & D3 & D4 & D5 \\
a & 145 & 223 & 346 & 78 & 89 \\
abandon & 4 & 0 & 0 & 5 & 3 \\
ability & 5 & 10 & 0 & 4 & 7 \\
able & 31 & 35 & 64 & 3 & 5 \\
about & 64 & 68 & 89 & 24 & 9 \\
above & 4 & 5 & 8 & 0 & 0 \\
abroad & 0 & 0 & 1 & 0 & 0 \\
absence & 2 & 4 & 0 & 0 & 0 \\
absent & 0 & 0 & 1 & 0 & 0 \\
absolute & 3 & 1 & 5 & 0 & 1 \\
abstract & 5 & 1 & 2 & 1 & 0 \\
abuse & 0 & 1 & 0 & 0 & 0 \\
academic & 1 & 3 & 0 & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

The meaning of the word is represented by documents it tends to occur in.
distributed semantics = vector semantics

Words as rows of term-term or word-word or word-context matrix: instead of documents we can define smaller contextes

sugar, a sliced lemon, a tablespoonful of apricot preserve or jam, a pinch each of
  left context: tablespoonful of
  right context: preserve or
  context/window size : 2

Shorter windows can capture more syntactic connections between words while larger windows can capture more semantic information.
distributed semantics = vector semantics

Modification:

- **TF-IDF:** \( w_{ij} = tf_{ij}idf_i \)
  - \( tf_{ij} \) frequency of the i-th term in the j-th document
  - \( idf_i \) inverse document frequency of the i-th term

\[ idf_i = \log \left( \frac{N}{df_i} \right) \]

- **Positive Pointwise Mutual Information:**

\[
PPMI(w, c) = \max(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0)
\]
distributed semantics = vector semantics

How do we quantify similarity of words?

Most commonly used metric is cosine:

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \|\mathbf{w}\|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

Alternatives:

- Jaccard($\mathbf{v}$, $\mathbf{w}$) $= \frac{\sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} \max(v_i, w_i)}$
- Dice($\mathbf{v}$, $\mathbf{w}$) $= \frac{2 \times \sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} (v_i + w_i)}$
- JS($\mathbf{v}$||$\mathbf{w}$) $= D(\mathbf{v} | \frac{v + w}{2}) + D(\mathbf{w} | \frac{v + w}{2})$
distributed semantics = vector semantics

Still holds:

- the size of word vectors is equal to the vocabulary size
- word vectors are sparse
word embedding

**Embedded representations:** short dense vectors that keep word semantics

**Embedding:** the whole process

Two approaches:

- count-based methods
- predictive methods

*Don't count, predict!* by Marco Baroni, Georgiana Dinu and German Kruszewski, ACL 2014.
language modeling

**Task:** Predict a word after the sequence of $n$ words.

**Classical approach:** maxim likelihood principle
maximize probability $P(w_n|w_{n-1}w_{n-2}...w_1)$

**2-gram approach:**

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$P(w_nw_{n-1}w_{n-2}...w_1) = P(w_1)P(w_2|w_1)P(w_3|w_2w_1)...P(w_n|w_{n-1}w_{n-2}...w_1)$$

$$\sim P(w_1)P(w_2|w_1)P(w_3|w_2)...P(w_n|w_{n-1})$$

**Issues:** out-of-vocabulary words, smoothing
neural network language model

**Task:** predict $w_t$ after a sequence of words $w_{t-3} w_{t-2} w_{t-1}$

**Feedforward NN classifier**

**Input:** word indexes  
**Output:** index of the next word

$V$: vocabulary size  
$d$: embedded word size
neural network language model

Network:
1) transform word $w_i$ to one hot representation $x_i$
2) $e = (Ex_1, Ex_2, \ldots, Ex_k) \leftarrow$ learned embedding matrix
3) $h = \text{activation}(We + b)$
4) $z = Uh$
5) $y = \text{softmax}(z)$

$\text{softmax}$ calculates probability distribution

Training: backpropagation
Loss: categorical cross entropy
Optimisation algorithm: stochastic gradient descent

word2vec approach

- **Inspection**: most of the complexity comes from the connection of the projection layer and the hidden layer as projections are dense.
- feedforward neural networks without hidden layer:
  - input layer
  - projection layer
  - output layer
- increase the amount of training data
- the plan is to use the **embedding matrix**, not to predict words as it might be expected.
Continuous Bag of Words (CBOW)

**Task:**
build a log-linear classifier that can correctly classify middle word for the given context
Continuous Bag of Words (CBOW)

- input: context words
  - one-hot representations

- output: hierarchical softmax
  - vocabulary is presented as Huffman binary tree

- projection layer:
  - it is shared as well as projection matrix - projected values are averaged
  - the order of words in the history does not influence the projection

**complexity per training example:** $N \times D + D \times \log_2(V)$

$N$ - context size, $D$ - embeddings dimensions, $V$ - vocabulary size
hierarchical softmax

- Softmax:
  \[
  \text{softmax}(x) = \left( \frac{e^{x_1}}{\sum_{i=1}^{C} e^{x_i}}, \ldots, \frac{e^{x_C}}{\sum_{i=1}^{C} e^{x_i}} \right)
  \]

- Computation complexity is \(O(V) \leftarrow c = V\)

- Vocabulary is presented as a Huffman binary tree

- **Hierarchical Softmax:**
  decompose calculating the probability of one word into a sequence of probability calculations

- Computational complexity \(O(\log_2(V))\)
Continuous skip-gram model

- **Task:** build a log-linear classifier that can correctly classify neighbour words for the given center word

- Precisely: for the center word and a given new word network will give us the probability of “a new word is a neighbour word” property
Continuous skip-gram model

- input: word
  - one-hot representations

- output: hierarchical softmax
  - vocabulary is presented as Huffman binary tree

- projection layer:
  - used for word embeddings

**Complexity per training example:** $C \times (D + D \times \log_2(V))$

*Complexity*:
- $C$ - maximum distance of the words,
- $D$ - embeddings dimensions,
- $V$ - vocabulary size
Continuous skip-gram model

This model learns two matrices: **embedding** matrix and **context** matrix.

For center word $w_j$, we compute:

$$p(w_k|w_j) = \frac{\exp(c_k \cdot v_j)}{\sum_{i \in |V|} \exp(c_i \cdot v_j)}$$
neural network training

- For every positive sample, we use some number of “negative” samples: samples we would like the network to predict value 0 for example: quick, sheep
- Some recommended values for the number of negative samples 5 to 20
- “unigram” table with frequencies
neural network training

- Backpropagation
- Stochastic gradient descent
  - start learning rate 0.025 and decrease it linearly
- Large scale parallel training of models is done by distributed framework called DistBelief
evaluation of word embeddings quality

Question:
"What is the word that is similar to X in the same sense as Y is similar to Z?"

Result is obtained by simple algebraic operations:
vector closest to the vector(Z)−vector(Y) + vector(X)

For example:
X = small, Y = big, Z = bigger
r = vector("bigger")−vector("big") + vector("small")
search for embedding that is closest (cosine metric) to r gives “smaller”
evaluation of word embeddings quality

- 5 types of semantic questions
- 9 types of syntactic questions

<table>
<thead>
<tr>
<th>Type of relationship</th>
<th>Word Pair 1</th>
<th>Word Pair 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common capital city</td>
<td>Athens</td>
<td>Greece</td>
</tr>
<tr>
<td>All capital cities</td>
<td>Astana</td>
<td>Kazakhstan</td>
</tr>
<tr>
<td>Currency</td>
<td>Angola</td>
<td>kwanza</td>
</tr>
<tr>
<td>City-in-state</td>
<td>Chicago</td>
<td>Illinois</td>
</tr>
<tr>
<td>Man-Woman</td>
<td>brother</td>
<td>sister</td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td>apparently</td>
<td>apparently</td>
</tr>
<tr>
<td>Opposite</td>
<td>possibly</td>
<td>impossibly</td>
</tr>
<tr>
<td>Comparative</td>
<td>great</td>
<td>greater</td>
</tr>
<tr>
<td>Superlative</td>
<td>easy</td>
<td>easiest</td>
</tr>
<tr>
<td>Present Participle</td>
<td>think</td>
<td>thinking</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>Switzerland</td>
<td>Swiss</td>
</tr>
<tr>
<td>Past tense</td>
<td>walking</td>
<td>walked</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>mouse</td>
<td>mice</td>
</tr>
<tr>
<td>Plural verbs</td>
<td>work</td>
<td>works</td>
</tr>
<tr>
<td></td>
<td>rapid</td>
<td>ethical</td>
</tr>
<tr>
<td></td>
<td>unethical</td>
<td>tough</td>
</tr>
<tr>
<td></td>
<td>lucky</td>
<td>luckiest</td>
</tr>
<tr>
<td></td>
<td>read</td>
<td>reading</td>
</tr>
<tr>
<td></td>
<td>swimming</td>
<td>Cambodian</td>
</tr>
<tr>
<td></td>
<td>dollar</td>
<td>swam</td>
</tr>
<tr>
<td></td>
<td>speak</td>
<td>dollars</td>
</tr>
</tbody>
</table>

In total: 8869 semantic and 10675 syntactic questions
evaluation of word embeddings quality

Vectors of various sizes on various dataset are learned and evaluated.

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimensionality</th>
<th>Training words</th>
<th>Accuracy [%]</th>
<th>Training time [days]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Semantic</td>
<td>Syntactic</td>
</tr>
<tr>
<td>3 epoch CBOV</td>
<td>300</td>
<td>783M</td>
<td>15.5</td>
<td>53.1</td>
</tr>
<tr>
<td>3 epoch Skip-gram</td>
<td>300</td>
<td>783M</td>
<td>50.0</td>
<td>55.9</td>
</tr>
<tr>
<td>1 epoch CBOV</td>
<td>300</td>
<td>783M</td>
<td>13.8</td>
<td>49.9</td>
</tr>
<tr>
<td>1 epoch CBOV</td>
<td>300</td>
<td>1.6B</td>
<td>16.1</td>
<td>52.6</td>
</tr>
<tr>
<td>1 epoch CBOV</td>
<td>600</td>
<td>783M</td>
<td>15.4</td>
<td>53.3</td>
</tr>
<tr>
<td>1 epoch Skip-gram</td>
<td>300</td>
<td>783M</td>
<td>45.6</td>
<td>52.2</td>
</tr>
<tr>
<td>1 epoch Skip-gram</td>
<td>300</td>
<td>1.6B</td>
<td>52.2</td>
<td>55.1</td>
</tr>
<tr>
<td>1 epoch Skip-gram</td>
<td>600</td>
<td>783M</td>
<td>56.7</td>
<td>54.5</td>
</tr>
</tbody>
</table>
properties of word embeddings

embedding(‘kings’) - embedding(‘king’) + embedding(‘queen’) → embedding(‘queens’)
properties of word embeddings

Examples of learned relationships:

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>
improvements

● **phrases** are included:
  ○ New York, Montreal Canadiens, ...
  ○ In total: 3 million of new words

● **data-driven approach is used for phrases extraction**
  ○ \((pab - \text{min\_count}) / (pa \times pb)\) where \(pa\), \(pb\), and \(pab\) are the number of occurrences of words \(a\), \(b\), and their combination
  ○ \(\text{min\_count}\) is a predefined value used for elimination of very infrequent phrases

● **phrases are treated as individual tokens**
  ○ New_York, Montreal_Canadiens, ...
improvements

● subsampling of frequent words
  ○ some words appear more often than other words but do not contribute to semantic
    the, a, in, …
  ○ exclusion of these words will improve the balance of rare and frequent words as well as speed
    up the training (according to results from 2x to 10x)

● \( z(w_i) \) is the fraction of total words in the corpus that are \( w_i \)
  for example, if \( peanut \) occurs 1,000 times in a 1 billion word corpus, then \( z(‘peanut’) = 1E-6 \)

● The probability of keeping the word \( w_i \)
  \[
P(w_i) = \left(\frac{\sqrt{z(w_i)}}{0.001} + 1\right) \cdot \frac{0.001}{z(w_i)}
  \]

● \( P(w_i) = 1 \) when \( z(w_i) <= 0.0026 \) → words which represent more than 0.26% of the total
  words will be subsampled.
comparison

- The size of embeddings: 300
- NEG-k: negative sampling with k negative samples per positive sample
- NSE: Noise Contrastive Estimation
- HS-Huffman: Hierarchical Softmax

<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>
Thank you!