An overview of word2vec

Benjamin Wilson

Berlin ML Meetup, July 8 2014
Outline

1. Introduction
2. Background & Significance
3. Architecture
4. CBOW word representations
5. Model scalability
6. Applications
word2vec associates words to points in space

- word2vec associates words with points in space
- word meaning and relationships between words are encoded spatially
- learns from input texts
- developed by Mikolov, Sutskever, Chen, Corrado and Dean in 2013 at Google Research
Similar words are closer together

- spatial distance corresponds to word similarity
- words are close together ⇔ their "meanings" are similar
- notation: word $w \mapsto \text{vec}[w]$ its point in space, as a position vector.

\[ \text{e.g. } \text{vec}[\text{woman}] = (0.1, -1.3). \]
Word relationships are displacements

- the displacement (vector) between the points of two words represents the word relationship.
- same word relationship $\Rightarrow$ same vector

Source: Linguistic Regularities in Continuous Space Word Representations, Mikolov et al, 2013

e.g.

$$\text{vec}[\text{queen}] - \text{vec}[\text{king}] = \text{vec}[\text{woman}] - \text{vec}[\text{man}]$$
What’s in a name?

How can a machine learn the meaning of a word? Machines only understand symbols!

Assume the Distributional Hypothesis (D.H.) (Harris, 1954):

“words are characterised by the company that they keep”

Suppose we read the word “cat”. What is the probability $P(w|\text{cat})$ that we’ll read the word $w$ nearby?

D.H. : the meaning of “cat” is captured by the probability distribution $P(\cdot|\text{cat})$. 
word2vec as shallow learning

- word2vec is a successful example of “shallow” learning
- word2vec can be trained as a very simple neural network
  - single hidden layer with no non-linearities
  - no unsupervised pre-training of layers (i.e. no deep learning)
- word2vec demonstrates that, for vectorial representations of words, shallow learning can give great results.
word2vec focuses on vectorization

- word2vec builds on existing research
- architecture is essentially that of Minh and Hinton’s log-bilinear model
- change of focus: vectorization, not language modelling.
word2vec scales

- word2vec scales very well, allowing models to be trained using *more data*.
- training speeded up by employing one of:
  - hierarchical softmax (more on this later)
  - negative sampling (for another day)
- runs on a single machine - can train a model at home
- implementation is published
Learning from text

- word2vec learns from input text
- considers each word $w_0$ in turn, along with its context $C$
- context = neighbouring words (here, for simplicity, 2 words forward and back)

<table>
<thead>
<tr>
<th>sample #</th>
<th>$w_0$</th>
<th>context $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>once</td>
<td>{upon, a}</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td>{upon, a, in, a}</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Two approaches: CBO Williams and Skip-gram

word2vec can learn the word vectors via two distinct learning tasks, CBO Williams and Skip-gram.

- CBO Williams: predict the current word $w_0$ given only $C$
- Skip-gram: predict words from $C$ given $w_0$
- Skip-gram produces better word vectors for infrequent words
- CBO Williams is faster by a factor of window size – more appropriate for larger corpora
- We will speak only of CBO Williams (life is short).
CBOW learning task

- Given only the current context $\mathcal{C}$, e.g.

$$\mathcal{C} = \{\text{upon, a, in, a}\}$$

predict which of all possible words is the current word $w_0$, e.g. $w_0 = \text{time}$.

- multiclass classification on the vocabulary $\mathcal{W}$

- output is $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathcal{C}) = P(\cdot | \mathcal{C})$ is a probability distribution on $\mathcal{W}$, e.g.

  $\hat{\mathbf{y}}$  
  "cat"  
  "trumpet"  
  "time"  
  \[ \begin{array}{c}
  \hat{y}\\
  0.11\\
  0.04\\
  0.3\\
  \end{array} \]

- train so that $\hat{\mathbf{y}}$ approximates target distribution $\mathbf{y}$ – “one-hot” on the current word, e.g.

  $\mathbf{y}$  
  "cat"  
  "trumpet"  
  "time"  
  \[ \begin{array}{c}
  y\\
  1.0\\
  0.0\\
  0.0\\
  \end{array} \]
Architecture

training CBOW with softmax regression

Model:

\[ \hat{y} = P(\cdot|C; \alpha, \beta) = \text{softmax}_\beta \left( \sum_{w \in C} \alpha_w \right), \]

where \( \alpha, \beta \) are families of parameter vectors. Pictorially:
stochastic gradient descent

- learn the model parameters (here, the linear transforms)
- minimize the difference between output distribution $\hat{y}$ and target distribution $y$, measured using the cross-entropy $H$:

$$H(y, \hat{y}) = - \sum_{w \in W} y_w \log \hat{y}_w$$

- given $y$ is one-hot, same as maximizing the probability of the correct outcome

$$\hat{y}_{w_0} = P(w_0|c; \alpha, \beta).$$

- use stochastic gradient descent: for each (current word, context) pair, update all the parameters once.
Post-training, associate every word $w \in W$ with a vector $\text{vec}[w]$:

- $\text{vec}[w]$ is the vector of synaptic strengths connecting the input layer unit $w$ to the hidden layer.

- more meaningfully, $\text{vec}[w]$ is the hidden-layer representation of the single-word context $C = \{w\}$.

- vectors are (artificially) normed to unit length (Euclidean norm), post-training.
Consider words $w, w' \in W$:

$$w \approx w' \iff P(\cdot | w) \approx P(\cdot | w')$$

(by the Distributional Hypothesis)

$$\iff \text{softmax}_\beta(\text{vec}[w]) \approx \text{softmax}_\beta(\text{vec}[w'])$$

(if model is well-trained)

$$\iff \text{vec}[w] \approx \text{vec}[w']$$

The last equivalence is tricky to show ...
We compare output distributions using the cross-entropy:

$$H(\text{softmax}_\beta(u), \text{softmax}_\beta(v))$$

- $\iff$ follows from continuity in $u, v$
- $\implies$ can be argued for from the convexity in $v$ when $u$ is fixed.
word relationship encoding

Given two examples of a single word relationship e.g.

*queen* is to *king* as *aunt* is to *uncle*

Find the closest point to

\[
\text{vec}[\text{queen}] + (\text{vec}[\text{uncle}] - \text{vec}[\text{aunt}]).
\]

It should be \text{vec}[\text{king}].

- Perform this test for many word relationship examples.
- CBOW & Skip-gram give correct answer in 58% - 69% of cases.
- Cosine distance is used (justified empirically!).
- What is the natural metric?

Source: *Efficient estimation of word representations in vector space*, Mikolov et al., 2013
softmax implementations are slow

Updates *all* second-layer parameters for every (current word, context) pair \((w_0, C)\) – very costly.

**Softmax models**

\[
P(w_0|C; \alpha, \beta) = \frac{\exp^{\beta^T v}}{\sum_{w' \in W} \exp^{\beta^T v'}}
\]

where \(v = \sum_{w \in C} \alpha_w\), and

\[
(\alpha_w)_{w \in W} \quad (\beta_w)_{w \in W}
\]

are the model parameters.

For each \((w_0, C)\) pair, must update \(O(|W|) \approx 100k\) parameters.
alternative models with fewer parameter updates

word2vec offers two alternatives to replace softmax.

- “hierarchical softmax” (H.S.) (Morin & Bengio, 2005)
- “negative sampling”, an adaptation of “noise contrastive estimation” (Gutmann & Hyvärinen, 2012) (skipped today)

- negative sampling scales better in vocabulary size
- quality of word vectors comparable
- both make significantly fewer parameter updates in the second-layer (no less parameters).
choose an arbitrary binary tree (# leaves = vocabulary size)
then \( P(\cdot|C) \) induces a weighting of the edges
think of each parent node \( n \) as a Bernoulli distribution \( P_n \) on its children. Then e.g.

\[
P(\text{time}|C) = P_{n_0}(\text{left}|C)P_{n_1}(\text{right}|C)P_{n_2}(\text{left}|C)
\]
choose an arbitrary binary tree (# leaves = vocabulary size)

then $P(\cdot|C)$ induces a weighting of the edges

think of each parent node $n$ as a Bernoulli distribution $P_n$ on its children. Then e.g.

$$P(\text{time}|C) = P_{n_0}(\text{left}|C)P_{n_1}(\text{right}|C)P_{n_2}(\text{left}|C)$$
Hierarchical softmax

- Choose an arbitrary binary tree (# leaves = vocabulary size)
- Then $P(\cdot|C)$ induces a weighting of the edges
- Think of each parent node $n$ as a Bernoulli distribution $P_n$ on its children. Then e.g.

$$P(\text{time}|C) = P_{n_0}(\text{left}|C)P_{n_1}(\text{right}|C)P_{n_2}(\text{left}|C)$$
choose an arbitrary binary tree (# leaves = vocabulary size)
then $P(\cdot|C)$ induces a weighting of the edges
think of each parent node $n$ as a Bernoulli distribution $P_n$ on its children. Then e.g.

$$P(\text{time}|C) = P_{n_0}(\text{left}|C)P_{n_1}(\text{right}|C)P_{n_2}(\text{left}|C)$$
In H.S., the probability of any outcome depends on only $O(\log |W|)$ parameters.

For each parent node $n$, assume that its Bernoulli distribution is modelled by: $P_n(\text{left}|C) = \sigma(\theta_n^{\top}v)$ where:

- $\theta_n$ is a vector of parameters
- $v = \sum_{w \in C} \alpha_w$ is the context vector
- $\sigma$ is the sigmoid function $\sigma(z) = \frac{1}{1+\exp^{-z}}$

Then, e.g.

\[
P(\text{time}|C) = \sigma(\theta_{n_0}^{\top}v) \cdot (1 - \sigma(\theta_{n_1}^{\top}v)) \cdot \sigma(\theta_{n_2}^{\top}v)
\]

depends on only $3 = \log_2 |W|$ of the parameter vectors $\theta_n$. 
word2vec H.S. uses Huffman tree

- could use any binary tree (# leaves = vocabulary size)
- word2vec uses a Huffman tree
  - frequent words have shorter paths in the tree
  - results in an even faster implementation

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>fat</td>
<td>3</td>
</tr>
<tr>
<td>fridge</td>
<td>2</td>
</tr>
<tr>
<td>zebra</td>
<td>1</td>
</tr>
<tr>
<td>potato</td>
<td>3</td>
</tr>
<tr>
<td>and</td>
<td>14</td>
</tr>
<tr>
<td>in</td>
<td>7</td>
</tr>
<tr>
<td>today</td>
<td>4</td>
</tr>
<tr>
<td>kangaroo</td>
<td>2</td>
</tr>
</tbody>
</table>
application to machine translation

- train word representations for e.g. English and Spanish separately
- the word vectors are similarly arranged!
- learn a linear transform that (approximately) maps the word vectors of English to the word vectors of their translations in Spanish
- same transform for all vectors

Source: Exploiting Similarities among Languages for Machine Translation, Mikolov, Quoc, Sutskever, 2013
English - Spanish: can guess the correct translation in 33% - 35% percent of the cases.

Source: *Exploiting Similarities among Languages for Machine Translation*, Mikolov, Quoc, Sutskever, 2013