Constructing and Evaluating Word Embeddings

Dr Marek Rei and Dr Ekaterina Kochmar
Computer Laboratory
University of Cambridge
Let's represent words (or any objects) as vectors. We want to construct them so that similar words have similar vectors.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>I live in Cambridge</td>
<td>19</td>
</tr>
<tr>
<td>I live in Paris</td>
<td>68</td>
</tr>
<tr>
<td>I live in Tallinn</td>
<td>0</td>
</tr>
<tr>
<td>I live in yellow</td>
<td>0</td>
</tr>
</tbody>
</table>

- Tallinn
- Cambridge
- London
- Paris
- yellow
- red
- blue
- green
Representing words as vectors

Let’s represent words (or any objects) as vectors. We want to construct them, so that similar words have similar vectors.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>I live in Cambridge</td>
<td>19</td>
</tr>
<tr>
<td>I live in Paris</td>
<td>68</td>
</tr>
<tr>
<td>I live in Tallinn</td>
<td>0</td>
</tr>
<tr>
<td>I live in yellow</td>
<td>0</td>
</tr>
</tbody>
</table>

- Tallinn
- Cambridge
- London
- Paris
- yellow
- red
- blue
- green
**1-hot vectors**

How can we represent words as vectors?

**Option 1**: each element represents a different word. Also known as “1-hot” or “1-of-V” representation.

<table>
<thead>
<tr>
<th></th>
<th>bear</th>
<th>cat</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>bear</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cat</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>frog</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

bear=[1.0, 0.0, 0.0]  
cat=[0.0, 1.0, 0.0]
1-hot vectors

When using 1-hot vectors, we can’t fit many and they tell us very little.

Need a separate dimension for every word we want to represent.
Distributed vectors

Option 2: each element represents a property, and they are shared between the words.

Also known as “distributed” representation.

<table>
<thead>
<tr>
<th></th>
<th>furry</th>
<th>dangerous</th>
<th>mammal</th>
</tr>
</thead>
<tbody>
<tr>
<td>bear</td>
<td>0.9</td>
<td>0.85</td>
<td>1</td>
</tr>
<tr>
<td>cat</td>
<td>0.85</td>
<td>0.15</td>
<td>1</td>
</tr>
<tr>
<td>frog</td>
<td>0</td>
<td>0.05</td>
<td>0</td>
</tr>
</tbody>
</table>

bear = [0.9, 0.85, 1.0]    cat = [0.85, 0.15, 1.0]
Distributed vectors group similar words/objects together.
Distributed vectors

Can use cosine to calculate similarity between two words

\[
\cos(a, b) = \frac{\sum a_i \cdot b_i}{\sqrt{\sum a_i^2} \cdot \sqrt{\sum b_i^2}}
\]

\[
\cos(\text{lion, bear}) = 0.998
\]

\[
\cos(\text{lion, dog}) = 0.809
\]

\[
\cos(\text{cobra, dog}) = 0.727
\]

Can use cosine to calculate similarity between two words
Distributed vectors

We can infer some information, based only on the vector of the word.

We don’t even need to know the labels on the vector elements.
Distributional hypothesis

Words which are similar in meaning occur in similar contexts.
(Harris, 1954)

You shall know a word by the company it keeps
(Firth, 1957)

He is reading a magazine
I was reading a newspaper

This magazine published my story
The newspaper published an article

She buys a magazine every month
He buys this newspaper every day
Count-based vectors

One way of creating a vector for a word:
Let’s count how often a word occurs together with specific other words.

He is reading a magazine
I was reading a newspaper

This magazine published my story
The newspaper published an article

She buys a magazine every month
He buys this newspaper every day

<table>
<thead>
<tr>
<th></th>
<th>reading</th>
<th>a</th>
<th>this</th>
<th>published</th>
<th>my</th>
<th>buys</th>
<th>the</th>
<th>an</th>
<th>every</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>magazine</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>newspaper</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Count-based vectors

- More frequent words dominate the vectors. Can use a weighting scheme like PMI or TF-IDF.

\[ pmi(x, z) = \log \frac{p(x, z)}{p(x)p(z)} \quad \text{tf-idf}(w, z) = freq_w,z \cdot \log \frac{V}{n_z} \]

- Large number of sparse features
  Can use matrix decomposition like Singular Value Decomposition (SVD) or Latent Dirichlet Allocation (LDA).
Neural networks will automatically try to discover useful features in the data, given a specific task.

**Idea:** Let’s allocate a number of parameters for each word and allow the neural network to automatically learn what the useful values should be.

Often referred to as “**word embeddings**”, as we are embedding the words into a real-valued low-dimensional space.
Embeddings through language modelling

Predict the next word in a sequence, based on the previous words.

Use this to guide the training for word embeddings.

Embeddings through error detection

Take a grammatically correct sentence and create a corrupted counterpart.

Train the neural network to assign a higher score to the correct version of each sentence.

Collobert et. al. 2011. *Natural Language Processing (Almost) from Scratch.*

my cat *climbed* a tree

my cat *bridge* a tree
Embedding matrix

Two ways of thinking about the embedding matrix.

1. **Each row** contains a word embedding, which we need to extract

2. It is a normal **weight matrix**, working with a 1-hot input vector
Word2vec

A popular tool for creating word embeddings.

Available from: [https://code.google.com/archive/p/word2vec/](https://code.google.com/archive/p/word2vec/)

Preprocess the data!

- Tokenise
- Lowercase (usually)

```
./word2vec -train input.txt -output vectors.txt -cbow 0 -size 100 -window 5 -negative 5 -hs 0 -sample 1e-3 -threads 8
```
Continuous Bag-of-Words (CBOW) model

Predict the current word, based on the surrounding words

Skip-gram model

Predict the surrounding words, based on the current word.

## Word similarity

<table>
<thead>
<tr>
<th>France</th>
<th>Jesus</th>
<th>Xbox</th>
<th>Reddish</th>
<th>Scratched</th>
<th>Megabits</th>
</tr>
</thead>
<tbody>
<tr>
<td>454</td>
<td>1973</td>
<td>6909</td>
<td>11724</td>
<td>29869</td>
<td>87025</td>
</tr>
<tr>
<td>Austria</td>
<td>God</td>
<td>Amiga</td>
<td>Greenish</td>
<td>Nailed</td>
<td>Octets</td>
</tr>
<tr>
<td>Belgium</td>
<td>Sati</td>
<td>Playstation</td>
<td>Bluish</td>
<td>Smashed</td>
<td>Mb/s</td>
</tr>
<tr>
<td>Germany</td>
<td>Christ</td>
<td>MSX</td>
<td>Pinkish</td>
<td>Punched</td>
<td>Bit/s</td>
</tr>
<tr>
<td>Italy</td>
<td>Satan</td>
<td>iPod</td>
<td>Purplish</td>
<td>Popped</td>
<td>Baud</td>
</tr>
<tr>
<td>Greece</td>
<td>Kali</td>
<td>Sega</td>
<td>Brownish</td>
<td>Crimped</td>
<td>Carats</td>
</tr>
<tr>
<td>Sweden</td>
<td>Indra</td>
<td>PSNUMBER</td>
<td>Greyish</td>
<td>Scraped</td>
<td>Kbit/s</td>
</tr>
<tr>
<td>Norway</td>
<td>Vishnu</td>
<td>HD</td>
<td>Grayish</td>
<td>Screwed</td>
<td>Megahertz</td>
</tr>
<tr>
<td>Europe</td>
<td>Ananda</td>
<td>Dreamcast</td>
<td>Whitish</td>
<td>Sectioned</td>
<td>Megapixels</td>
</tr>
<tr>
<td>Hungary</td>
<td>Parvati</td>
<td>GeForce</td>
<td>Silvery</td>
<td>Slashed</td>
<td>Gbit/s</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Grace</td>
<td>Capcom</td>
<td>Yellowish</td>
<td>Ripped</td>
<td>Amperes</td>
</tr>
</tbody>
</table>

Collobert et. al. 2011. *Natural Language Processing (Almost) from Scratch.*
Word similarity
The task of analogy recovery. Questions in the form:

\[ a \text{ is to } b \text{ as } c \text{ is to } d \]

The system is given words \( a, b, c \), and it needs to find \( d \). For example:

- ‘apple’ is to ‘apples’ as ‘car’ is to ?
- or
- ‘man’ is to ‘woman’ as ‘king’ is to ?

**Analogy recovery**

**Task:** $a$ is to $b$ as $c$ is to $d$

**Idea:** The direction of the relation should remain the same.

\[ a - b \approx c - d \]

\[ \text{man} - \text{woman} \approx \text{king} - \text{queen} \]

\[ d_w = \arg\max_{d'_w \in V} (\cos(a - b, c - d')) \]
Analogy recovery

Task: a is to b as c is to d

Idea: The offset of vectors should reflect their relation.

\[ a - b \approx c - d \]
\[ d \approx c - a + b \]

\[ \text{queen} \approx \text{king} - \text{man} + \text{woman} \]

\[ d_w = \arg \max_{d'_w \in V} (\cos(d'_w, c - a + b)) \]
## Analogy recovery

<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>

Example output using word2vec vectors.
Word embeddings in practice

Word2vec is often used for pretraining.

- It will help your models start from an informed position
- Requires only plain text - which we have a lot
- Is very fast and easy to use
- Already pretrained vectors also available (trained on 100B words)

However, for best performance it is important to continue training (fine-tuning).

Raw word2vec vectors are good for predicting the surrounding words, but not necessarily for your specific task.

Simply treat the embeddings the same as other parameters in your model and keep updating them during training.
Word embeddings in practice

Word embeddings are the building blocks for higher-level models
Questions?
Count-based vs neural

Comparison of count-based and neural word vectors on 5 types of tasks and 14 different datasets:

1. Semantic relatedness
2. Synonym detection
3. Concept categorization
4. Selectional preferences
5. Analogy recovery

Baroni et. al. 2014. Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors.
Some of these conclusions are challenged by:
Levy et. al. 2015. *Improving Distributional Similarity with Lessons Learned from Word Embeddings.*
Count-based vs neural

The best parameter choices for counting models:

- window size 2 (bigger is not always better)
- weighted with PMI, not LMI
- no dimensionality reduction (not using SVD or NNMF)

The best parameters for the neural network model:

- window size 5
- negative sampling (not hierarchical softmax)
- subsampling of frequent words
- dimensionality 400