Distributional Semantics: Word Embeddings

Slide credits: Yoav Goldberg (mostly), Omer Levy, Ido Dagan
Words as Vectors

We can arrange the words in a huge, sparse matrix, where each row is a word, and each column is a context.
Dimensionality reduction

- Vector spaces often range from tens of thousands to millions of context dimensions
- Some of the methods to reduce dimensionality:
  - Select context features based on various relevance criteria
  - Random indexing
  - Following claimed to also have a beneficial *smoothing* effect:
    - Singular Value Decomposition
    - Non-negative matrix factorization
    - Probabilistic Latent Semantic Analysis
    - Latent Dirichlet Allocation
Words as Vectors

We often apply SVD or similar technique of dimensionality reduction.
Words as Vectors – It works

Nearest neighbours to dog

- cat
- horse
- fox
- pet
- rabbit
- pig
- animal
- mongrel
- sheep
- pigeon
Words as Vectors – It works
Nearest neighbours to dog

2-word window

- cat
- horse
- fox
- pet
- rabbit
- pig
- animal
- mongrel
- sheep
- pigeon
Words as Vectors – It works
Nearest neighbours to **dog**

<table>
<thead>
<tr>
<th>2-word window</th>
<th>30-word window</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>kennel</td>
</tr>
<tr>
<td>horse</td>
<td>puppy</td>
</tr>
<tr>
<td>fox</td>
<td>pet</td>
</tr>
<tr>
<td>pet</td>
<td>bitch</td>
</tr>
<tr>
<td>rabbit</td>
<td>terrier</td>
</tr>
<tr>
<td>pig</td>
<td>rottweiler</td>
</tr>
<tr>
<td>animal</td>
<td>canine</td>
</tr>
<tr>
<td>mongrel</td>
<td>cat</td>
</tr>
<tr>
<td>sheep</td>
<td>to bark</td>
</tr>
<tr>
<td>pigeon</td>
<td>Alastian</td>
</tr>
</tbody>
</table>
word2vec

Tool for computing continuous distributed representations of words.

**Introduction**

This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing applications and for further research.

**Quick start**

- Download the code: svn checkout http://word2vec.googlecode.com/svn/trunk/
- Run 'make' to compile word2vec tool
- Run the demo scripts: ./demo-word.sh and ./demo-phrases.sh
- For questions about the toolkit, see http://groups.google.com/group/word2vec-toolkit

**How does it work**

The word2vec tool takes a text corpus as input and produces the word vectors as output. It first learns a vocabulary from the training text by estimating the probability distribution over words. Then it computes continuous representations of words.
Google

Web  Images  Videos  News  Maps  More  Search tools

About 384 results (0.56 seconds)

MLMU.cz - Radim Řehůřek - Word2vec & friends (7.1.2015 ...  
www.youtube.com/watch?v=wTp3P2UnTfQ  
Jan 14, 2015 - Uploaded by Marek Modrý  
I'll go over a particular model published by Google, called word2vec, its optimizations, applications and ...

Word2Vec convergence on Vimeo  
https://vimeo.com/112168934  
Nov 18, 2014  
This is "Word2Vec convergence" by MaciejLyst on Vimeo, the home for high quality videos and the people who ...

Statistical Semantic入門~分布仮説からword2vecまで #1 ...  
www.ustream.tv/recorded/43497190  
Statistical Semantic入門~分布仮説からword2vecまで #1. February 5, 2014 at 7:16pm ...

Statistical Semantic入門~分布仮説からword2vecまで #2, PFI ...  
www.ustream.tv/recorded/43497424  
Feb 5, 2014  
非常に説明がわかりやすいです！「ゲーミフィケーション入門」と「マーケティングとスタートアップの話」を見ました。どちらも非常に理解しやすかった ...

GigaOM Show: Samsung watch secrets spilled! B&N's Nook ...  
https://www.picsrico.com/528e406e7f89a233b7b7
word2vecによる自然言語処理

Tomas Mikolovらによって提案されたニューラルネットワーク（CBOW, Skip-gram）のオープンソース実装word2vecについて、基本的な使い方を体験し、さらにその仕組みを学ぶ書籍です。

基本的な使い方から、自分の好きなコーパスの作り方、登録の背景、仕組み、さらには応用例や弱点についてもコンパクトなボリュームで概観できます。付録にはword2vecの出力結果を主成分分析を使って可視化する方法について解説しています。

なお本書はEbook版のみの販売となります。

著者の西尾先生による本書の解説[リンク]
From Distributional to Distributed Semantics

This part of the talk

- `word2vec` as a black box
- a peek inside the black box
- relation between word-embeddings and the distributional representation
- tailoring word embeddings to your needs using `word2vec沣`
word2vec

Tool for computing continuous distributed representations of words.

**Introduction**

This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing applications and for further research.

**Quick start**

- Download the code: svn checkout http://word2vec.googlecode.com/svn/trunk/
- Run `make` to compile word2vec tool
- Run the demo scripts: `./demo-word.sh` and `./demo-phrases.sh`
- For questions about the toolkit, see http://groups.google.com/group/word2vec-toolkit

**How does it work**

The word2vec tool takes a text corpus as input and produces the word vectors as output. It first

...
word2vec

feed in text

Wikipedia

wait a few hours

dog = (0.12, -0.32, 0.92, 0.43, -0.3 ...)
cat = (0.15, -0.29, 0.90, 0.39, -0.32 ...)
chair = (0.8, 0.9, -0.76, 0.29, 0.52 ...)

get a $|V| \times d$ matrix $W$ where each row is a vector for a word
- dog
  - cat, dogs, dachshund, rabbit, puppy, poodle, rottweiler, mixed-breed, doberman, pig
- sheep
  - cattle, goats, cows, chickens, sheeps, hogs, donkeys, herds, shorthorn, livestock
- november
  - october, december, april, june, february, july, september, january, august, march
- jerusalem
  - tiberias, jaffa, haifa, israel, palestine, nablus, damascus, katamon, ramla, safed
- teva
  - pfizer, schering-plough, novartis, astrazeneca, glaxosmithkline, sanofi-aventis, mylan, sanofi, genzyme, pharmacia
Working with Dense Vectors

Word Similarity

- Similarity is calculated using cosine similarity:

\[ \text{sim}(\text{dog}, \text{cat}) = \frac{\text{dog} \cdot \text{cat}}{\|\text{dog}\| \|\text{cat}\|} \]

- For normalized vectors (\(\|x\| = 1\)), this is equivalent to a dot product:

\[ \text{sim}(\text{dog}, \text{cat}) = \text{dog} \cdot \text{cat} \]

- Normalize the vectors when loading them.
Working with Dense Vectors

Finding the most similar words to $\vec{dog}$

- Compute the similarity from word $\vec{v}$ to all other words.
Working with Dense Vectors

Finding the most similar words to $\mathbf{dog}$

- Compute the similarity from word $\mathbf{v}$ to all other words.
- This is a **single matrix-vector product**: $W \cdot \mathbf{v}^\top$

![Diagram showing matrix-vector product](image)
Working with Dense Vectors

Finding the most similar words to $\vec{d}$

- Compute the similarity from word $\vec{v}$ to all other words.
- This is a **single matrix-vector product**: $W \cdot \vec{v}^T$

```
<table>
<thead>
<tr>
<th>V</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cat</td>
</tr>
<tr>
<td></td>
<td>chair</td>
</tr>
<tr>
<td></td>
<td>june</td>
</tr>
<tr>
<td></td>
<td>sun</td>
</tr>
<tr>
<td></td>
<td>bark</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>eat</td>
</tr>
</tbody>
</table>

\[ W \cdot \vec{v}^T = \begin{bmatrix} 0.9 & -0.3 & -0.1 & -0.9 & 0.3 & \cdots & \cdots & 0.2 \end{bmatrix} \]
```

- Result is a $|V|$ sized vector of similarities.
- Take the indices of the $k$-highest values.
Working with Dense Vectors

Finding the most similar words to \( \text{dog} \)

- Compute the similarity from word \( \vec{v} \) to all other words.
- This is a **single matrix-vector product**: \( W \cdot \vec{v}^T \)

\[
\begin{align*}
\text{cat} & \quad \text{chair} \\
\text{june} & \quad \text{sun} \\
\text{bark} & \quad \ldots \ldots \\
\text{eat} & \quad \ldots \ldots \\
\end{align*}
\]

\[
W \cdot \vec{v}^T = \begin{bmatrix}
0.9 & -0.3 & -0.1 & -0.9 & 0.3 & 0.2 \\
\end{bmatrix}
\]

- Result is a \( |V| \) sized vector of similarities.
- Take the indices of the \( k \)-highest values.
- **FAST!** for 180k words, \( d=300 \): \( \sim30\text{ms} \)
Most Similar Words, in python+numpy code

```python
W, words = load_and_normalize_vectors("vecs.txt")
# W and words are numpy arrays.
w2i = {w:i for i,w in enumerate(words)}

dog = W[w2i[‘dog’]]  # get the dog vector

sims = W.dot(dog)  # compute similarities

most_similar_ids = sims.argsort()[-1:-10:-1]
sim_words = words[most_similar_ids]
```
Working with Dense Vectors

Similarity to a group of words

- “Find me words most similar to cat, dog and cow”.
- Calculate the pairwise similarities and sum them:
  \[ W \cdot \vec{cat} + W \cdot \vec{dog} + W \cdot \vec{cow} \]
- Now find the indices of the highest values as before.
Working with Dense Vectors

Similarity to a group of words

- “Find me words most similar to cat, dog and cow”.
- Calculate the pairwise similarities and sum them:
  \[ W \cdot \text{cat} + W \cdot \text{dog} + W \cdot \text{cow} \]
- Now find the indices of the highest values as before.
- Matrix-vector products are wasteful. **Better option:**
  \[ W \cdot (\text{cat} + \text{dog} + \text{cow}) \]
Working with dense word vectors can be very efficient.
Working with dense word vectors can be very efficient.

But where do these vectors come from?
How does word2vec work?

word2vec implements several different algorithms:

Two training methods

- Negative Sampling
- Hierarchical Softmax

Two context representations

- Continuous Bag of Words (CBOW)
- Skip-grams
How does word2vec work?

word2vec implements several different algorithms:

Two training methods

- Negative Sampling
- Hierarchical Softmax

Two context representations

- Continuous Bag of Words (CBOW)
- Skip-grams

We’ll focus on skip-grams with negative sampling.

Intuitions apply for other models as well.
How does word2vec work?

- Represent each word as a $d$ dimensional vector.
- Represent each context as a $d$ dimensional vector.
- Initialize all vectors to random weights.
- Arrange vectors in two matrices, $W$ and $C$. 

$$
\begin{align*}
\text{words} & : |V_w| \\
\text{contexts} & : |V_c|
\end{align*}
$$
How does word2vec work?

While more text:

- **Extract a word window:**
  
  A springer is [ a cow or heifer close to calving ].

  \[
  c_1 \quad c_2 \quad c_3 \quad w \quad c_4 \quad c_5 \quad c_6
  \]

  - \( w \) is the focus word vector (row in \( W \)).
  - \( c_i \) are the context word vectors (rows in \( C \)).
How does word2vec work?

While more text:

- Extract a word window:
  
  \[
  \text{A springer is} \left[ \text{a cow or heifer close to calving} \right].
  \]

\[
\begin{align*}
  &c_1 &c_2 &c_3 &w &c_4 &c_5 &c_6
\end{align*}
\]

- Try setting the vector values such that:

\[
\sigma(w \cdot c_1) + \sigma(w \cdot c_2) + \sigma(w \cdot c_3) + \sigma(w \cdot c_4) + \sigma(w \cdot c_5) + \sigma(w \cdot c_6)
\]

is high
How does word2vec work?

While more text:

- Extract a word window:
  A springer is [ a cow or heifer close to calving ].
  \[ c_1 \quad c_2 \quad c_3 \quad w \quad c_4 \quad c_5 \quad c_6 \]

- Try setting the vector values such that:
  \[ \sigma(w \cdot c_1) + \sigma(w \cdot c_2) + \sigma(w \cdot c_3) + \sigma(w \cdot c_4) + \sigma(w \cdot c_5) + \sigma(w \cdot c_6) \]

  is **high**

- Create a corrupt example by choosing a random word \( w' \)
  [ a cow or comet close to calving ]
  \[ c_1 \quad c_2 \quad c_3 \quad w' \quad c_4 \quad c_5 \quad c_6 \]

- Try setting the vector values such that:
  \[ \sigma(w' \cdot c_1) + \sigma(w' \cdot c_2) + \sigma(w' \cdot c_3) + \sigma(w' \cdot c_4) + \sigma(w' \cdot c_5) + \sigma(w' \cdot c_6) \]

  is **low**
Word2vec Formulation

\( D \) : a set of original (correct) word-context pairs

\( \bar{D} \) : a set of corrupt (incorrect) word-context pairs

- \textit{k negative samples} are generated for each correct sample

The model needs to estimate:

\[ P(D=1|w,c) : \text{the probability that } (w,c) \text{ is from } D \]

- Should be high for pairs from \( D \), low if

\[ P(D=0|w,c) = 1 - P(D=1|w,c) : \quad \bar{D} \]

the probability that \((w,c)\) is from
Word2vec Formulation (cont.)

Modeling probability as sigmoid of dot product score $s(w,c)$:

$$ P(D = 1|w, c) = \frac{1}{1 + e^{-s(w,c)}} $$

Learning goal: find vectors $w, c$ for all words and contexts that maximize log-likelihood of data $D \cup \tilde{D}$:

$$ \mathcal{L}(\Theta; D, \tilde{D}) = \sum_{(w,c) \in D} \log P(D = 1|w, c) + \sum_{(w,c) \in \tilde{D}} \log P(D = 0|w, c) $$

Negative samples generated by original frequency, or smoothed - deemphasizing frequent words, better in practice:

$$ \frac{\#(w)}{\sum_{w'} \#(w')} \quad \text{or} \quad \frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0.75}} $$
The *Skip-gram* model assumes independence between the context elements.

Denote \( c_{1:k} \) a context of \( k \) elements:

\[
P(D = 1 | w, c_i) = \frac{1}{1 + e^{-w \cdot c_i}}
\]

\[
P(D = 1 | w, c_{1:k}) = \prod_{i=1}^{k} P(D = 1 | w, c_i) = \prod_{i=1}^{k} \frac{1}{1 + e^{-w \cdot c_i}}
\]

\[
\log P(D = 1 | w, c_{1:k}) = \log \sum_{i=1}^{k} \frac{1}{1 + e^{-w \cdot c_i}}
\]

Very effective in practice, commonly used.
How does word2vec work?

The training procedure results in:
- \( w \cdot c \) for **good** word-context pairs is **high**.
- \( w \cdot c \) for **bad** word-context pairs is **low**.
- \( w \cdot c \) for **ok-ish** word-context pairs is **neither high nor low**.

As a result:
- Words that share many contexts get close to each other.
- Contexts that share many words get close to each other.

At the end, word2vec throws away \( C \) and returns \( W \).
Reinterpretation

Imagine we didn’t throw away $C$. Consider the product $WC^T$. 
Reinterpretation

Imagine we didn’t throw away $C$. Consider the product $WC^T$

The result is a matrix $M$ in which:

- Each row corresponds to a word.
- Each column corresponds to a context.
- Each cell correspond to $w \cdot c$, an association measure between a word and a context.
Reinterpretation

\[ W C^T = M \]

Does this remind you of something?
Reinterpretation

\[ W \cdot C^T = M \]

Does this remind you of something?

Very similar to SVD over distributional representation:

\[ U \cdot S \cdot V^T \]
What is SGNS learning?

- A $V_w \times V_c$ matrix
- Each cell describes the relation between a specific word-context pair

\[
\vec{w} \cdot \vec{c} = ?
\]

“Neural Word Embeddings as Implicit Matrix Factorization”
Levy & Goldberg, NIPS 2014
What is SGNS learning?

- We prove that for large enough $d$ and enough iterations
What is SGNS learning?

- We **prove** that for large enough $d$ and enough iterations
- We get the word-context PMI matrix

“Neural Word Embeddings as Implicit Matrix Factorization”
Levy & Goldberg, NIPS 2014
What is SGNS learning?

- We **prove** that for large enough $d$ and enough iterations
- We get the word-context PMI matrix, shifted by a global constant

\[
Opt(\overrightarrow{w} \cdot \overrightarrow{c}) = PMI(w, c) - \log k
\]
What is SGNS learning?

- SGNS is doing something very similar to the older approaches

- SGNS is factorizing the traditional word-context PMI matrix

- So does SVD!

- Do they capture the same similarity function?
<table>
<thead>
<tr>
<th>Target Word</th>
<th>SGNS</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>dog, rabbit, cats, poodle, pig</td>
<td>dog, rabbit, pet, monkey, pig</td>
</tr>
</tbody>
</table>
### SGNS vs SVD

<table>
<thead>
<tr>
<th>Target Word</th>
<th>SGNS</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>wine</td>
<td>wines, grape</td>
<td>wines</td>
</tr>
<tr>
<td></td>
<td>grapes</td>
<td>grape</td>
</tr>
<tr>
<td></td>
<td>winemaking</td>
<td>grapes</td>
</tr>
<tr>
<td></td>
<td>tasting</td>
<td>varietal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vintages</td>
</tr>
<tr>
<td>Target Word</td>
<td>SGNS</td>
<td>SVD</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>November</td>
<td>October December April January July</td>
<td>October December April June March</td>
</tr>
</tbody>
</table>
But *word2vec* is still better, isn’t it?

- Plenty of evidence that *word2vec* outperforms traditional methods
  - In particular: “Don’t count, predict!” (Baroni et al., 2014)

- How does this fit with our story?
The Big Impact of “Small” Hyperparameters
Hyperparameters

- **word2vec** is more than just an algorithm...

- Introduces many **engineering tweaks** and **hyperparameter settings**
  - May seem minor, but **make a big difference** in practice
  - Their impact is often more significant than the embedding algorithm’s

- These modifications can be ported to distributional methods!

Levy, Goldberg, Dagan (In submission)
Hyperparameters

• Preprocessing
• Association Metric
• Postprocessing
Hyperparameters

- Preprocessing
- **Association Metric**
- Postprocessing

Levy, Goldberg, Dagan (In submission)
Association Metric Hyperparameters

• Since SGNS and PMI are strongly related, we can import 2 of SGNS’s hyperparameters to traditional PMI:
  1. Shifted PMI
  2. Negative Sampling Smoothing

• Both stem from the negative sampling procedure

Levy, Goldberg, Dagan (In submission)
Negative Sampling Smoothing

- Recall that SGNS picks $w' \sim P$ to form negative $(w', c)$ examples

- Our analysis assumes $P$ is the unigram distribution

$$P(w) = \frac{\#w}{\sum_{w' \in V_w} \#w'}$$

Levy, Goldberg, Dagan (In submission)
Negative Sampling Smoothing

- Recall that SGNS picks $w' \sim P$ to form negative $(w', c)$ examples

- Our analysis assumes $P$ is the unigram distribution

- In practice, it’s a **smoothed** unigram distribution

\[
P^{0.75}(w) = \frac{(\#w)^{0.75}}{\sum_{w' \in V_w}(\#w')^{0.75}}
\]

- This little change makes a big difference

Levy, Goldberg, Dagan (In submission)
Negative Sampling Smoothing

• This smoothing has an analogue in PMI

• Replace $P(w)$ with $P^{0.75}(w)$:

$$PMI^{0.75}(w, c) = \log \frac{P(w, c)}{P^{0.75}(w)P(c)}$$

• Yields a **dramatic** improvement with **every method** on **every task**

Levy, Goldberg, Dagan (In submission)
Experiments & Results

• We compared several methods, while controlling for hyperparameters
  • PPMI, SVD(PPMI), SGNS, GloVe

• Methods perform on-par in most tasks
  • Slight advantage to SVD in word similarity
  • SGNS is better at syntactic analogies
  • SGNS is robust in general

• Negative sampling smoothing accounts for much of the differences observed in “Don’t count, predict!”

Levy, Goldberg, Dagan (In submission)
Other Hyperparameters

• There are many other hyperparameters that can be investigated

• Perhaps the most interesting one is the type of context
What’s in a Context?
What’s in a Context?

• Importing ideas from embeddings improves distributional methods

• Can distributional ideas also improve embeddings?

• Idea: change SGNS’s default BoW contexts into dependency contexts

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Example

Australian scientist discovers star with telescope

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Target Word

Australian scientist **discovering** star with telescope

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Bag of Words (BoW) Context

Australian scientist discovers star with telescope

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Bag of Words (BoW) Context

Australian scientist discovers star with telescope

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Bag of Words (BoW) Context

Australian scientist discovers star with telescope

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Syntactic Dependency Context

Australian scientist discovers star with telescope

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Syntactic Dependency Context

Australian **scientist** discovers **star** **telescope**

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Syntactic Dependency Context

Australian scientist discovers star telescope

"Dependency-Based Word Embeddings"
Levy & Goldberg, ACL 2014
# Embedding Similarity with Different Contexts

<table>
<thead>
<tr>
<th>Target Word</th>
<th>Bag of Words (k=5)</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry Potter’s school</td>
<td>Dumbledore, hallows, half-blood, Malfoy, Snape</td>
<td>Sunnydale, Collinwood, Calarts, Greendale, Millfield</td>
</tr>
</tbody>
</table>

Related to
Harry Potter

Schools

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
## Embedding Similarity with Different Contexts

<table>
<thead>
<tr>
<th>Target Word</th>
<th>Bag of Words (k=5)</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing (computer scientist)</td>
<td>nondeterministic, non-deterministic, computability, deterministic, finite-state</td>
<td>Pauling, Hotelling, Heting, Lessing, Hamming</td>
</tr>
</tbody>
</table>

**Related to computability**

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
## Embedding Similarity with Different Contexts

<table>
<thead>
<tr>
<th>Target Word</th>
<th>Bag of Words (k=5)</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>dancing</td>
<td>singing</td>
<td>singing</td>
</tr>
<tr>
<td>(dance gerund)</td>
<td>dance</td>
<td>rapping</td>
</tr>
<tr>
<td></td>
<td>dances</td>
<td>breakdancing</td>
</tr>
<tr>
<td></td>
<td>dancers</td>
<td>miming</td>
</tr>
<tr>
<td></td>
<td>tap-dancing</td>
<td>busking</td>
</tr>
</tbody>
</table>

**Related to dance**

**Gerunds**

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
What is the effect of different context types?

- Thoroughly studied in distributional methods
  - Lin (1998), Padó and Lapata (2007), and many others...

**General Conclusion:**

- Bag-of-words contexts induce *topical* similarities
- Dependency contexts induce *functional* similarities
  - Share the same semantic type
  - Cohyponyms

- Holds for *embeddings* as well

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Peeking into Skip-Gram’s Black Box

• In explicit representations, we can look at the features and analyze

• But embeddings are a black box!
• Dimensions are latent and don’t necessarily have any meaning

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Peeking into Skip-Gram’s Black Box

• Skip-Gram allows a peek...

• Contexts are embedded in the same space!

• Given a word \( w \), find the contexts \( c \) it “activates” most:

\[
\arg \max_c (\vec{w} \cdot \vec{c})
\]

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
## Associated Contexts

<table>
<thead>
<tr>
<th>Target Word</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hogwarts</td>
<td>students/prep_at&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>educated/prep_at&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>student/prep_at&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>stay/prep_at&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>learned/prep_at&lt;sup&gt;-1&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
## Associated Contexts

<table>
<thead>
<tr>
<th>Target Word</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing</td>
<td>machine/nn$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>test/nn$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>theorem/poss$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>machines/nn$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>tests/nn$^{-1}$</td>
</tr>
</tbody>
</table>

“Dependency-Based Word Embeddings”  
Levy & Goldberg, ACL 2014
## Associated Contexts

<table>
<thead>
<tr>
<th>Target Word</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>dancing</td>
<td>dancing/conj</td>
</tr>
<tr>
<td></td>
<td>dancing/conj$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>singing/conj$^{-1}$</td>
</tr>
<tr>
<td></td>
<td>singing/conj</td>
</tr>
<tr>
<td></td>
<td>ballroom/nn</td>
</tr>
</tbody>
</table>

“Dependency-Based Word Embeddings”
Levy & Goldberg, ACL 2014
Context matters

Choose the correct contexts for your application

- larger window sizes – more topical
- dependency relations – more functional
Context matters

Choose the correct contexts for your application

- larger window sizes – more topical
- dependency relations – more functional
- only noun-adjective relations
- only verb-subject relations
Context matters

Choose the correct contexts for your application

- larger window sizes – more topical
- dependency relations – more functional
- only noun-adjective relations
- only verb-subject relations
- context: time of the current message
- context: user who wrote the message
Context matters

Choose the correct contexts for your application

- larger window sizes – more topical
- dependency relations – more functional
- only noun-adjective relations
- only verb-subject relations
- context: time of the current message
- context: user who wrote the message
- ...
- the sky is the limit
Software

word2vecf
https://bitbucket.org/yoavgo/word2vecf

- Extension of word2vec.
- Allows saving the context matrix.
- Allows using arbitrary contexts.
  - Input is a (large) file of word context pairs.
Software

**hyperwords**

[https://bitbucket.org/omerlevy/hyperwords/](https://bitbucket.org/omerlevy/hyperwords/)

- Python library for working with either sparse or dense word vectors (similarity, analogies).
- Scripts for creating dense representations using word2vecf or SVD.
- Scripts for creating sparse distributional representations.
Software

dissect
http://clic.cimec.unitn.it/composes/toolkit/

- Given vector representation of words...
- ...derive vector representation of phrases/sentences
- Implements various composition methods
Summary

Distributional Semantics

- Words in similar contexts have similar meanings.
- Represent a word by the contexts it appears in.
- But what is a context?

Neural Models (word2vec)

- Represent each word as dense, low-dimensional vector.
- Same intuitions as in distributional vector-space models.
- Efficient to run, scales well, modest memory requirement.
- Dense vectors are convenient to work with.
- Still helpful to think of the context types.

Software

- Build your own word representations.