



Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**



Distributional Semantics

Advanced Machine Learning for NLP

Jordan Boyd-Graber

SLIDES ADAPTED FROM YOAV GOLDBERG AND OMER LEVY

From Distributional to Distributed Semantics

The new kid on the block

- Deep learning / neural networks
- “Distributed” word representations
 - Feed text into neural-net. Get back “word embeddings”.
 - Each word is represented as a low-dimensional vector.
 - Vectors capture “semantics”
- `word2vec` (Mikolov et al)

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`

word2vec

 **tmikolov** / **word2vec**

 **Code**


 **Issues** **35**


 **Pull requests** **0**

 **Projects** **0**

 **Puls**

Automatically exported from code.google.com/p/word2vec

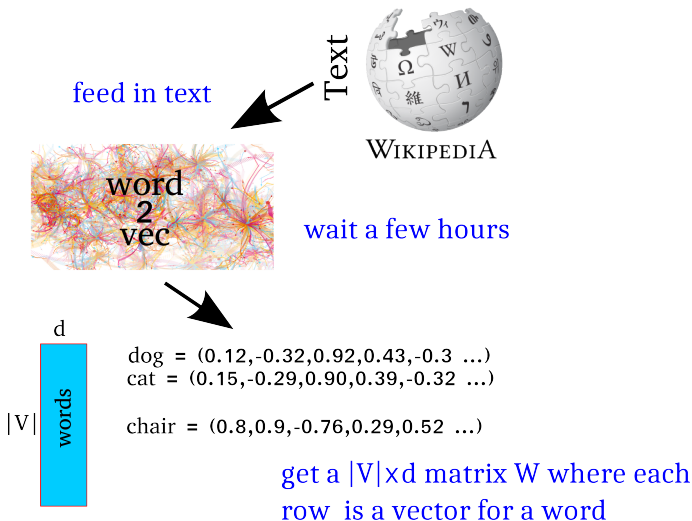
 **42 commits**

 **2 branches**

 **0 releases**

Branch: **master** ▼

New pull request



word2vec

- 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*:

$$\text{sim}(\vec{dog}, \vec{cat}) = \frac{\vec{dog} \cdot \vec{cat}}{\|\vec{dog}\| \|\vec{cat}\|}$$

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

$$\text{sim}(\vec{dog}, \vec{cat}) = \vec{dog} \cdot \vec{cat}$$

- **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 \vec{dog}

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

The diagram illustrates the matrix-vector product $W \cdot \vec{v}^T$ to find word similarities. It shows a matrix W of size $|V| \times d$ and a vector \vec{v}^T of size $d \times 1$. The result is a vector of similarities of size $1 \times |V|$.

The matrix W is represented by a blue vertical bar with the following words listed vertically: cat, chair, june, sun, bark, ..., ..., eat. The dimension d is indicated above the bar, and $|V|$ is indicated to the left.

The vector \vec{v}^T is represented by a small blue vertical bar with the word "dog" next to it. The dimension d is indicated below the bar.

The result is a horizontal bar with the following values: 0.9, -0.3, -0.1, -0.9, 0.3, ..., ..., 0.2. The dimension $1 \times |V|$ is indicated below the bar.

The equation is summarized as:

$$W \quad \vec{v}^T = \text{similarities}$$
$$|V| \times d \quad d \times 1 \quad 1 \times |V|$$

Working with Dense Vectors

Finding the most similar words to \vec{dog}

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

The diagram illustrates the matrix-vector product $W \cdot \vec{v}^T$ to find word similarities. On the left, a matrix W of size $|V| \times d$ is shown as a blue vertical rectangle. It contains the words: cat, chair, june, sun, bark, ..., ..., eat. Above the first row is the label 'd'. To the left of the matrix is the label $|V|$. In the center, a vector \vec{v}^T of size $d \times 1$ is shown as a small blue vertical rectangle, with the word 'dog' below it. To the right of the vector is an equals sign. Further right, a row of similarity values is shown in a pink horizontal rectangle: 0.9, -0.3, -0.1, -0.9, 0.3, ..., ..., 0.2. Below this row are the words: cat, chair, june, sun, bark, ..., ..., eat. Below the matrix and vector, the dimensions are specified: $|V| \times d$ for W , $d \times 1$ for \vec{v}^T , and $1 \times |V|$ for the result, which is labeled 'similarities'.

$$\begin{matrix} & d \\ \begin{matrix} |V| \\ \text{cat} \\ \text{chair} \\ \text{june} \\ \text{sun} \\ \text{bark} \\ \dots \\ \dots \\ \text{eat} \end{matrix} & \begin{matrix} \text{dog} \end{matrix} & = & \begin{matrix} 0.9 & -0.3 & -0.1 & -0.9 & 0.3 & \dots & \dots & 0.2 \end{matrix} \\ & & & \begin{matrix} \text{cat} \\ \text{chair} \\ \text{june} \\ \text{sun} \\ \text{bark} \\ \dots \\ \dots \\ \text{eat} \end{matrix} \end{matrix}$$
$$\begin{matrix} W & \vec{v}^T & = & \text{similarities} \\ |V| \times d & d \times 1 & & 1 \times |V| \end{matrix}$$

- 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 \vec{d}_{dog}

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

The diagram illustrates the matrix-vector product $W \cdot \vec{v}^T$ used to find word similarities. On the left, a matrix W of size $|V| \times d$ is shown, where $|V|$ is the number of words in the vocabulary and d is the dimension of the word vectors. The words listed are cat, chair, june, sun, bark, ..., ..., eat. A blue vertical bar highlights the column corresponding to the word 'dog', which is the vector \vec{v} of size $d \times 1$. This vector is multiplied by the matrix W to produce a vector of similarities of size $1 \times |V|$. The resulting vector is shown on the right, with values for each word: cat (0.9), chair (-0.3), june (-0.1), sun (-0.9), bark (0.3), ..., ..., eat (0.2). The word 'dog' is also listed below the vector, indicating it is the word being compared.

$$\begin{matrix} & d \\ & \text{cat} \\ & \text{chair} \\ & \text{june} \\ & \text{sun} \\ & \text{bark} \\ & \dots \\ & \dots \\ & \text{eat} \\ |V| & \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \\ W & \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \\ |V| \times d & \end{matrix} \quad \begin{matrix} & d \times 1 \\ & \text{dog} \\ & \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \\ \vec{v}^T & \end{matrix} = \begin{matrix} & 1 \times |V| \\ & \text{similarities} \\ & \begin{bmatrix} 0.9 & -0.3 & -0.1 & -0.9 & 0.3 & \dots & \dots & 0.2 \end{bmatrix} \\ & \end{matrix}$$

- Result is a $|V|$ sized vector of similarities.
- Take the indices of the k -highest values.
- **FAST!** for 180k words, $d=300$: $\sim 30\text{ms}$

Working with Dense Vectors

Most Similar Words, in python+numpy code

```
W, words = load_and_norm_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:

$$W \cdot \vec{cat} + W \cdot \vec{dog} + W \cdot \vec{cow}$$

- Now find the indices of the highest values as before.
- Matrix-vector products are wasteful. **Better option:**

$$W \cdot (\vec{cat} + \vec{dog} + \vec{cow})$$

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

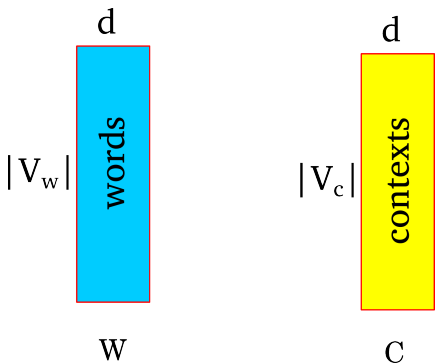
- Continuous Bag of Words (CBOW)
- **Skip-grams**

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 .



How does word2vec work?

While more text:

- Extract a word window:

A springer is[a cow or **heifer** close to calving].

c_1 c_2 c_3 w c_4 c_5 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:

A springer is[a cow or **heifer** close to calving].

c_1 c_2 c_3 w c_4 c_5 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**

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.

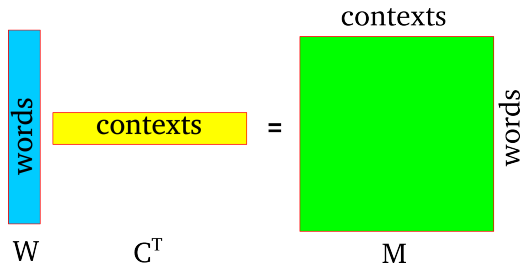
At the end, word2vec throws away C and returns W .

Reinterpretation

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

Reinterpretation

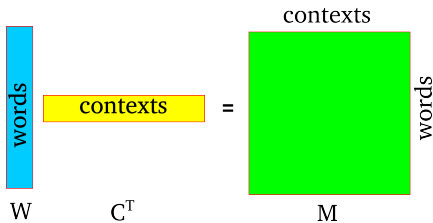
Imagine we didn't throw away C . Consider the product WC^T



The result is a matrix M in which:

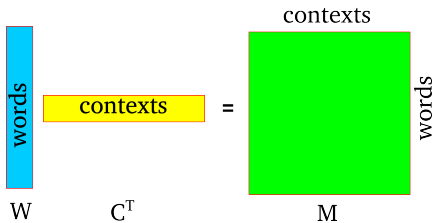
- Each row corresponds to a word.
- Each column corresponds to a context.
- Each cell: $w \cdot c$, association between word and context.

Reinterpretation



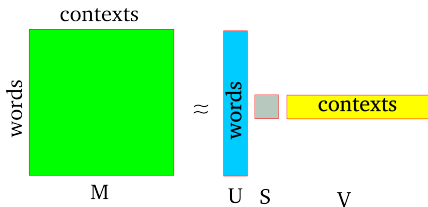
Does this remind you of something?

Reinterpretation



Does this remind you of something?

Very similar to SVD over distributional representation:



Relation between SVD and word2vec

SVD

- Begin with a word-context matrix.
- Approximate it with a product of low rank (thin) matrices.
- Use thin matrix as word representation.

word2vec (skip-grams, negative sampling)

- Learn thin word and context matrices.
- These matrices can be thought of as approximating an implicit word-context matrix.
 - Levy and Goldberg (NIPS 2014) show that this implicit matrix is related to the well-known PPMI matrix.

Relation between SVD and word2vec

word2vec is a dimensionality reduction technique over an (implicit) word-context matrix.

Just like SVD.

With few tricks (Levy, Goldberg and Dagan, TACL 2015) we can get SVD to perform just as well as `word2vec`.

Relation between SVD and word2vec

word2vec is a dimensionality reduction technique over an (implicit) word-context matrix.

Just like SVD.

With few tricks (Levy, Goldberg and Dagan, TACL 2015) we can get SVD to perform just as well as `word2vec`.

However, `word2vec`...

- ... **works without building / storing the actual matrix in memory.**
- ... **is very fast to train, can use multiple threads.**
- ... **can easily scale to huge data and very large word and context vocabularies.**