Distributional Semantics

Jordan Boyd-Graber

University of Maryland

Slides Adapted from Yoav Goldberg and Omer Levy

From Distributional to Distributed Semantics

The new kid on the block

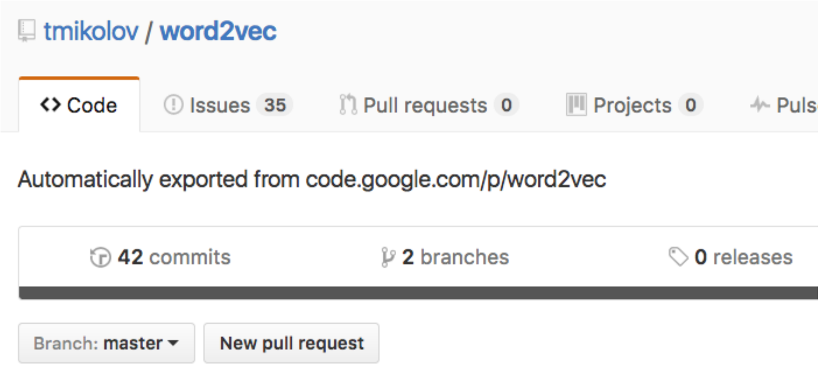
- Deep learning / neural networks
- "Distributed" word representations
  - Feed text into neural-net. Get back "word embeddings".
  - Each word is represented as a low-dimensional vector.
  - Vectors capture "semantics"
- `word2vec` (Mikolov et al)

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

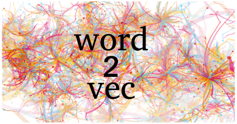

tmikolov / word2vec
Code
Issues 35
Pull requests 0
Projects 0
Automatically exported from code.google.com/p/word2vec
42 commits
2 branches
0 releases
Branch: master
New pull request

word2vec

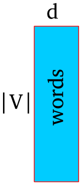
Text

WIKIPEDIA

feed in text



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|xd matrix W where each row is a vector for a word

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*:

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

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

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

- **Normalize the vectors when loading them.**

Working with Dense Vectors

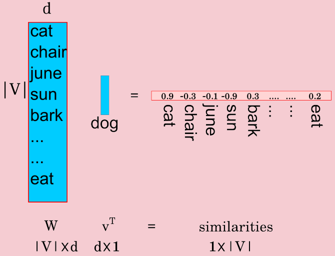
Finding the most similar words to $\vec{dog}$

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

Working with Dense Vectors

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

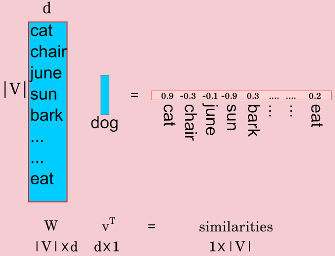
- Compute the similarity from word $\vec{v}$ to all other words.
- This is a **single matrix-vector product**: $W \cdot \vec{v}^\top$



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**: $W \cdot \vec{v}^{\top}$

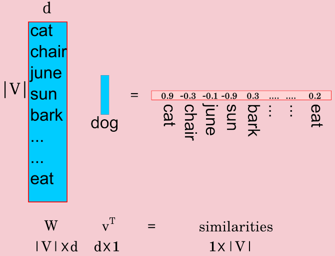


- Result is a $|V|$ sized vector of similarities.
- Take the indices of the k-highest values.

Working with Dense Vectors

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

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



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

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:**

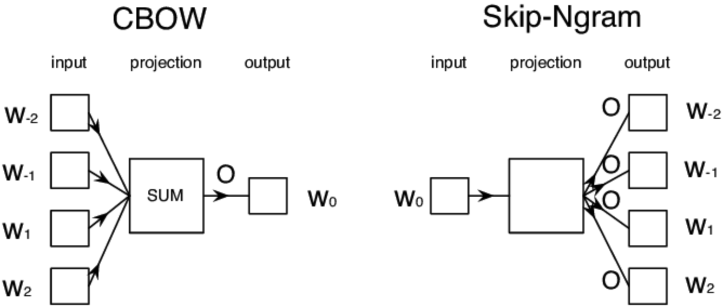
$$W \cdot (\vec{cat} + \vec{dog} + \vec{cow})$$

Working with dense word vectors can be very efficient.

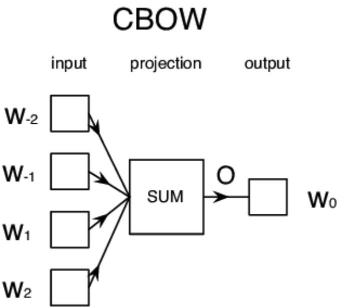
Working with dense word vectors can be very efficient.

But where do these vectors come from?

Flavors of word2vec



Flavors of word2vec



Two context representations

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

Flavors of word2vec

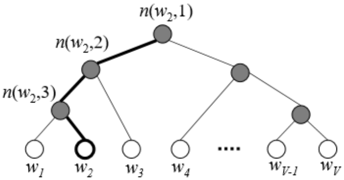
Two context representations

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

Flavors of word2vec

Two context representations

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



Flavors of word2vec

Two context representations

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

Two training methods

- Negative Sampling
- Hierarchical Softmax

Flavors of word2vec

Two context representations

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

Two training methods

- **Negative Sampling**
- Hierarchical Softmax

Flavors of word2vec

Two context representations

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

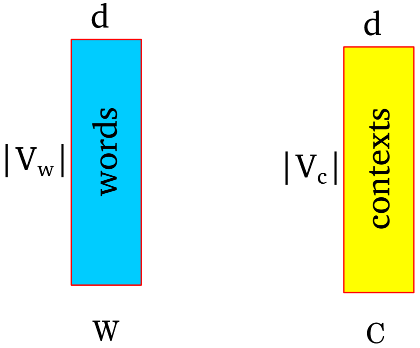
Two training methods

- Negative Sampling
- Hierarchical Softmax

But once you understand one, others follow.

How does word2vec work?

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



The Prediction Problem

$$p(c|w;\theta) = \frac{\exp v_c \cdot v_w}{\sum_{c' \in C} \exp v_{c'} \cdot v_w} \tag{1}$$

- Predict context word(s)
- from focus word
- Looks a lot like logistic regression!

Total objective function (in log space):

$$\arg\max_{\theta} \sum_{(w,c) \in D} \log p(c|w) = \sum_{(w,c) \in D} \left[\log \exp v_c \cdot v_w - \log \sum_{c'} \exp v_{c'} \cdot v_w \right] \tag{2}$$

The Prediction Problem

$$p(c \mid w; \theta) = \frac{\exp v_c \cdot v_w}{\sum_{c' \in C} \exp v_{c'} \cdot v_w} \tag{1}$$

- Predict context word(s)
- from focus word
- Looks a lot like logistic regression!

Total objective function (in log space):

$$\arg\max_{\theta} \sum_{(w,c) \in D} \log p(c \mid w) = \sum_{(w,c) \in D} \left[\log \exp v_c \cdot v_w - \log \sum_{c'} \exp v_{c'} \cdot v_w \right] \tag{2}$$

The Prediction Problem

$$p(c\,|\,w;\theta) = \frac{\exp v_c \cdot v_w}{\sum_{c' \in C} \exp v_{c'} \cdot v_w} \tag{1}$$

- Predict context word(s)
- from focus word
- Looks a lot like logistic regression!

Total objective function (in log space):

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log p(c\,|\,w) = \sum_{(w,c)\in D} \left[\log \exp v_c \cdot v_w - \log \sum_{c'} \exp v_{c'} \cdot v_w \right] \tag{2}$$

The Prediction Problem

$$p(c|w;\theta) = \frac{\exp v_c \cdot v_w}{\sum_{c' \in C} \exp v_{c'} \cdot v_w} \tag{1}$$

- Predict context word(s)
- from focus word
- Looks a lot like logistic regression!

Total objective function (in log space):

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log p(c|w) = \sum_{(w,c)\in D} \left[\log \exp v_c \cdot v_w - \log \sum_{c'} \exp v_{c'} \cdot v_w \right] \tag{2}$$

How does word2vec work?

While more text:

- Extract a word window:

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

$$\begin{array}{ccccccc} \texttt{a} & \texttt{cow} & \texttt{or} & \textbf{heifer} & \texttt{close} & \texttt{to} & \texttt{calving} \\ c_1 & c_2 & c_3 & w & c_4 & c_5 & c_6 \end{array}$$

- 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 ].`

$$\begin{array}{ccccccc} \texttt{a} & \texttt{cow} & \texttt{or} & \textbf{heifer} & \texttt{close} & \texttt{to} & \texttt{calving} \\ c_1 & c_2 & c_3 & w & c_4 & c_5 & c_6 \end{array}$$

- 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 ].
```

$$\begin{array}{ccccccc} \texttt{a} & \texttt{cow} & \texttt{or} & \textbf{heifer} & \texttt{close} & \texttt{to} & \texttt{calving} \\ c_1 & c_2 & c_3 & w & c_4 & c_5 & c_6 \end{array}$$

- 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' (negative sample) [a cow or **comet** close to calving]

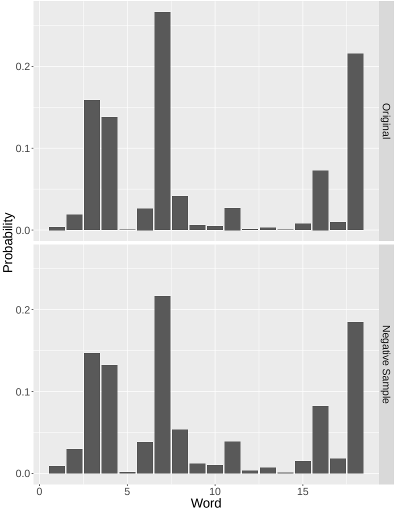
$$\begin{array}{ccccccc} \texttt{a} & \texttt{cow} & \texttt{or} & \textbf{comet} & \texttt{close} & \texttt{to} & \texttt{calving} \\ c_1 & c_2 & c_3 & w' & c_4 & c_5 & c_6 \end{array}$$

- 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**

Negative Sampling Distribution



$$p^{\mathrm{NS}}(w) = \frac{f(w)^{\frac{3}{4}}}{\sum_{w'} f(w)^{\frac{3}{4}}} \tag{3}$$

Brings down frequent terms, brings up infrequent terms

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.

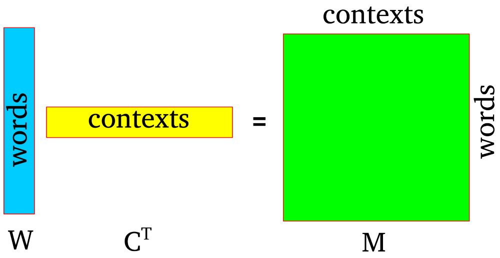
At the end, word2vec throws away C and returns W.

Reinterpretation

Imagine we didn't throw away C. Consider the product $WC^\top$

Reinterpretation

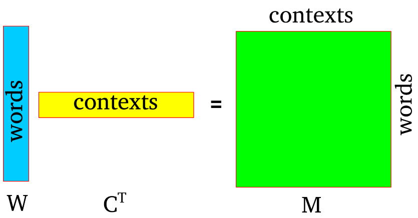
Imagine we didn't throw away C. Consider the product $WC^\top$



The result is a matrix M in which:

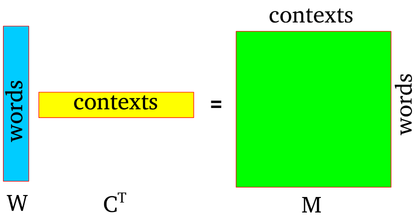
- Each row corresponds to a word.
- Each column corresponds to a context.
- Each cell: $w \cdot c$, association between word and context.

Reinterpretation



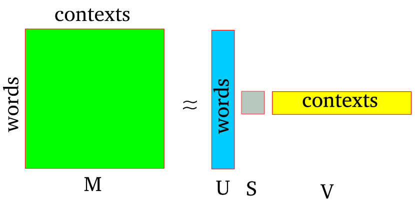
Does this remind you of something?

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.**